Capítulo 9 - Grupos de Dados Heterogêneos

Registros

Registros podem ser entendidos como grupos de dados, logicamente relacionados, podendo ser do mesmo tipo ou de tipos diferentes, agrupados ou não.

Registros ***físicos*** são unidades, representadas por conjuntos de ***bytes***, para armazenamento e manipulação do ponto de vista do meio de armazenamento (fita, disco etc.).

Registros ***lógicos*** são unidades, representadas por conjuntos de informações, para definição e organização do ponto de vista da abstração de dados.

Exemplo:

Supor que seja desejado representar uma ficha com campos a serem preenchidos com informações sobre uma pessoa, como a mostrada abaixo.

<table>
<tr><td colspan="4">NOME</td></tr>
<tr><td colspan="4">ENDEREÇO</td></tr>
<tr><td>RUA</td><td>NÚMERO</td><td>CIDADE</td><td>ESTADO</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td>ESTADO CIVIL</td><td>SEXO</td><td>IDADE</td><td>SALÁRIO</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
</table>

Individualmente, essas informações poderiam ser armazenadas em termos dos tipos básicos, como apresentado a seguir:

```
caracteres      NOME, RUA, CIDADE, ESTADO, CIVIL
inteiro         NUMERO, IDADE
caractere       SEXO
real            SALÁRIO
```

É comum, no entanto, que alguns destes dados representem uma mesma informação, por exemplo, o ENDEREÇO, como entidade, só é completa quando existirem os dados referentes a RUA, NÚMERO, CIDADE e ESTADO. Até mesmo a ficha inteira pode ser tratada como uma única entidade ou um conjunto de informações heterogêneas.

Definição de registros

Forma geral:

```
tipo <nome do registro>
= registro
    | <tipo 1> <lista de campos 1>
    | ...
    | <tipo N> <lista de campos N>
    fim registro
```

Exemplo:

Para o caso do ENDEREÇO, apresentado anteriormente, pode-se ter a estrutura de dados mostrada abaixo.

```
tipo ENDEREÇOS
=  registro
   |  caracteres        RUA
   |  inteiro           NÚMERO
   |  caracteres        CIDADE, ESTADO
   fim registro
```

Para o caso de PESSOA, pode-se ter uma composição de tipos, como a seguir.

```
tipo PESSOAS
=  registro
   |  caracteres NOME
   |  ENDEREÇOS      ENDEREÇO
   |  caracteres     CIVIL
   |  caractere      SEXO
   |  real           SALÁRIO
   fim registro
```

Pode-se, ainda, ter grupos homogêneos de registros, ou registros cujos campos fossem arranjos de qualquer tipo.

Exemplos:

```
tipo FICHÁRIO
= PESSOAS [10]

tipo GRUPO
=  registro
   |  inteiro      N
   |  FICHÁRIO P
   fim registro
```

Utilização de registros

- Indicação de campo

Forma geral:

<nome do registro>.<nome do campo>

Exemplo:

Supondo um registro de nome P, do tipo PESSOA, definido anteriormente, a indicação de cada campo pode ser feita como o indicado abaixo.

```
P.NOME
P.ENDEREÇO.RUA
P.ENDEREÇO.NÚMERO
P.ENDEREÇO.CIDADE
P.ENDEREÇO.ESTADO
P.CIVIL
P.SEXO
P.SALÁRIO
```

- Transferência em nível de campos

Forma geral:

<nome do registro>.<nome do campo> ← <valor>

Exemplo:

Supondo um registro de nome P, do tipo PESSOA, definido anteriormente, a atribuição a cada campo poderia ser feita como indicado abaixo.

```
P.NOME                  ← "José"
P.ENDEREÇO.RUA          ← "Rua 1"
P.ENDEREÇO.NÚMERO       ←  1
P.ENDEREÇO.CIDADE       ← "X"
P.ENDEREÇO.ESTADO       ← "Y"
P.CIVIL                 ← "solteiro"
P.SEXO                  ← "M"
P.SALÁRIO               ← 1.0
```

- Transferência a nível de registros completos

Forma geral:

```
<nome do registro> ← (<valor do campo 1>,
                      ...
                      <valor do campo N>)
```

Exemplo:

Supondo um registro de nome P, do tipo PESSOA, definido anteriormente, a atribuição de cada campo pode ser feita como o indicado abaixo.

```
P.ENDEREÇO ← ( "Rua 1", 1, "X", "Y" )

P ← ( "José",
      ("Rua 1", 1, "X", "Y"),
       "solteiro",
       "M",
       1.0)
```

- Transferência entre dispositivos e memória

Forma geral:

<dispositivo> ← <nome do registro>
<nome do registro> ← <dispositivo>

<dispositivo> ← <nome do registro>.<nome do campo>
<nome do registro>.<nome do campo> ← <dispositivo>

Observação:

Tendo em vista aplicações práticas, a segunda forma deve ser preferida em relação à primeira.

Exemplo:

Supondo um registro de nome P, do tipo PESSOA, definido anteriormente, a transferência de todo o conteúdo de um registro pode ser feita como indicado abaixo.

P ← teclado
tela ← P

ou melhor:

```
( P.NOME,
  P.ENDEREÇO.RUA,
  P.ENDEREÇO.NÚMERO,
  P.ENDEREÇO.CIDADE,
  P.ENDEREÇO.ESTADO,
  P.CIVIL,
  P.SEXO,
  P.SALÁRIO)  ←  teclado

tela ← ( P.NOME,
         P.ENDEREÇO.RUA,
         P.ENDEREÇO.NÚMERO,
         P.ENDEREÇO.CIDADE,
         P.ENDEREÇO.ESTADO,
         P.CIVIL,
         P.SEXO,
         P.SALÁRIO)
```

- Comparação (verificar a igualdade de registros)

Forma geral:

<nome do registro 1> = <nome do registro 2>

Exemplo:

Supondo dois registros de nome P1 e P2, do tipo PESSOA, definido anteriormente, a comparação pode ser feita como o indicado abaixo:

```
se P1 = P2 então
|   ! comandos !
fim se ! P1 = P2 !
```

Observação:

Dois registros serão considerados iguais somente se pertencerem ao mesmo tipo e se todos os campos tiverem conteúdos idênticos.

Caso pertençam a tipos diferentes, ainda poderá haver semelhança entre eles, desde que se atenda aos critérios de equivalência entre tipos.

- Abreviação

Forma geral:

```
com <registro>
|   ! comandos, utilizando somente os campos !
fim com
```

Exemplo:

Supondo dois registros de nome P1 e P2, do tipo PESSOA, definido anteriormente, a abreviação pode ser feita como o indicado abaixo:

```
com P1
|  NOME       ← P2.NOME
|  ENDEREÇO   ← P2.ENDEREÇO
|  CIVIL      ← "solteiro"
|  SEXO       ← "M"
|  SALARIO    ←  1.0
fim com ! P1 !
```

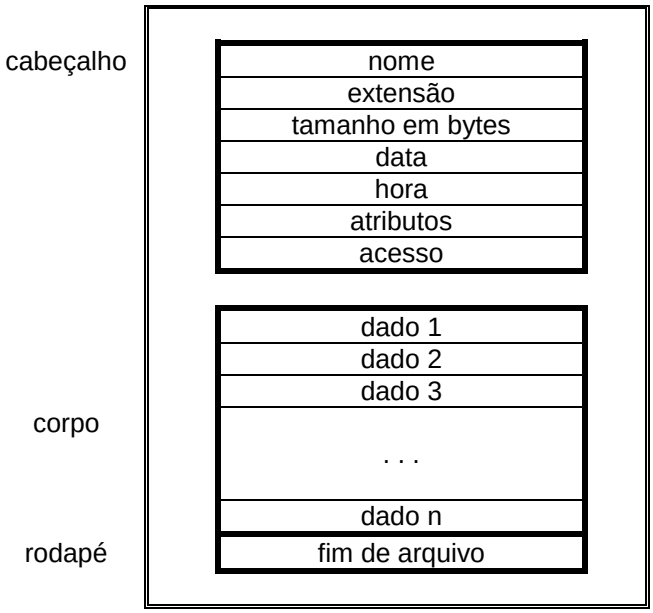
Observação:

Um nome de campo é definido no escopo da declaração de um registro. Outros registros, ou campos, ou variáveis, poderão ter o mesmo nome, desde que não haja ambigüidades quanto ao uso, como mostrado acima.

Arquivos

Pode-se chamar de ***arquivo*** um conjunto organizado de registros, armazenado em algum dispositivo de memória secundária.

Um arquivo poderá ter a seguinte organização para efeitos práticos:



Formato:

- cabeçalho - é um registro padrão contendo:

  - nome - nome externo do arquivo (no diretório ou índice)
  - tamanho - tamanho de arquivo em número de registros
  - data - data de criação ou de última alteração
  - hora - hora de criação ou de última alteração
  - atributos - informações sobre segurança, visibilidade, disponibilidade para leitura etc.
  - acesso - forma de organização:

    - ***sequencial***:
      os registros estão dispostos fisicamente um após o outro, segundo a ordem de gravação, e para se acessar um determinado registro, deve-se passar, obrigatoriamente, por todos os anteriores a ele. A cada operação, os dados em um registro são tratados e coloca-se o próximo registro à disposição, automaticamente;

    - ***direto***:
      o acesso aos registros pode ser feito fora da ordem, por meio da indicação de um índice. O índice natural de um registro é o seu número de ordem. A cada operação, um registro é localizado e tratado, mantendo-se o mesmo à disposição, até que outra operação determine o que deve ser feito;

    - ***indexado***:
      o mesmo que o anterior, só que o índice natural pode substituído pela combinação de um ou mais campos do registro (chamada de CHAVE ou ÍNDICE). Esse arquivo deve ser ordenado por essa combinação de campos do registro para facilitar o acesso (indexação).

- corpo/dados - é a sequência de armazenamento dos registros (corpo)

Definição de arquivos

Forma geral:

tipo <nome do tipo> = arquivo de <tipo>

onde:

<tipo> - indica o tipo de registro
<arquivos> - indica os nomes de variáveis desse tipo

Exemplos:

tipo ARQ = arquivo de inteiro
tipo ARQPESSOA = arquivo de PESSOA

ARQ A

ARQPESSOA P1, P2

Utilização de arquivos

- Abertura:

Forma geral:

abrir (<arquivo>, <nome externo>, <operação>)

onde:

<arquivo> - é um nome de variável
<nome externo> - é um nome válido para o armazenamento
<acesso> - é a forma de organização para acesso:
- seqüencial
- direto
- indexado

<operação> - é o tipo de operação:
- leitura
- gravação
- alteração

Observações:

A operação de abertura posiciona um arquivo no seu primeiro registro. No caso de gravação, dispensa-se todo o conteúdo existente.

Um arquivo só poderá ser aberto (ou fechado) uma única vez. Se for desejado mudar a operação, ele deverá ser fechado, primeiro, e nova abertura deverá ser feita.

O formato de armazenamento pode ser na forma de texto ou de dados binária.

- Fechamento:

Forma geral:

fechar (<arquivo>)

- Transferências:

Forma geral:

<arquivo> ← <registro>
<arquivo> (<índice>) ← <registro>

<registro> ← <arquivo>
<registro> ← <arquivo> (<índice>)

- Indexação ou posicionamento:

Forma geral:

<arquivo> (<índice>)

Observação:

O posicionamento só poderá ser feito em arquivos com tipo de acesso direto (ou indexado, mediante determinação de chave).

Procedimentos e funções associados a arquivos

- Função para teste de fim de arquivo (FDA):

lógico função FDA (<arquivo>)

Exemplo:

```
repetir enquanto não FDA ( A )
|  P ← A
fim repetir
```

- Função para indicar o número de registros:

inteiro função tamanho (<nome do arquivo>)

Exemplo:

```
repetir para ( X = 1: tamanho ( “DADOS” ) )
|  P ← A
fim repetir
```

- Função para indicar validade de índice:

lógico função válido (<nome do arquivo>, <índice>)

Observação:

Uma operação qualquer sobre arquivos de tipo direto, ou indexado, só poderá ser feita em posições válidas.

Exemplo:

```
se válido ( “DADOS”, NOME_PROCURADO )
|  P ←   A ( NOME_PROCURADO )
fim se
```

- Função para verificar a existência de um arquivo:

```
lógico função existir ( <nome do arquivo> )
```

Exemplo:

```
se existir ( “DADOS” ) então
|   P ← A ( NOME_PROCURADO )
fim se
```

Observação:

Uma operação de leitura, ou alteração, só poderá ocorrer em arquivos já existentes.
Uma operação de gravação cria um arquivo novo. Se já existir um, com o mesmo nome e descrição, todo o seu conteúdo será apagado.

- Função para indicar a posição atual em um arquivo:

```
inteiro função posição ( <arquivo> )
```

Exemplo:

```
tela ← posição ( A )
```

- Procedimento para indexação:

```
indexar ( <nome do arquivo> )
```

Exemplo:

```
indexar ( "DADOS" )
```

Observações:

Um arquivo só poderá ser tratado de forma indexada depois de ordenadas suas chaves de busca (indexação).

Não é prevista a manutenção automática da ordem em arquivos indexados. Sugere-se fazer a inserção e a remoção de registros com o cuidado de prover a necessária atualização da ordem.

Os índices, primariamente, serão organizados segundo os campos de registros, na ordem em que foram definidos e respeitando uma seqüência crescente alfanumérica.

- Procedimento para posicionar a partir do início do arquivo

```
posicionar ( <arquivo>, <posição> )
```

Observação:

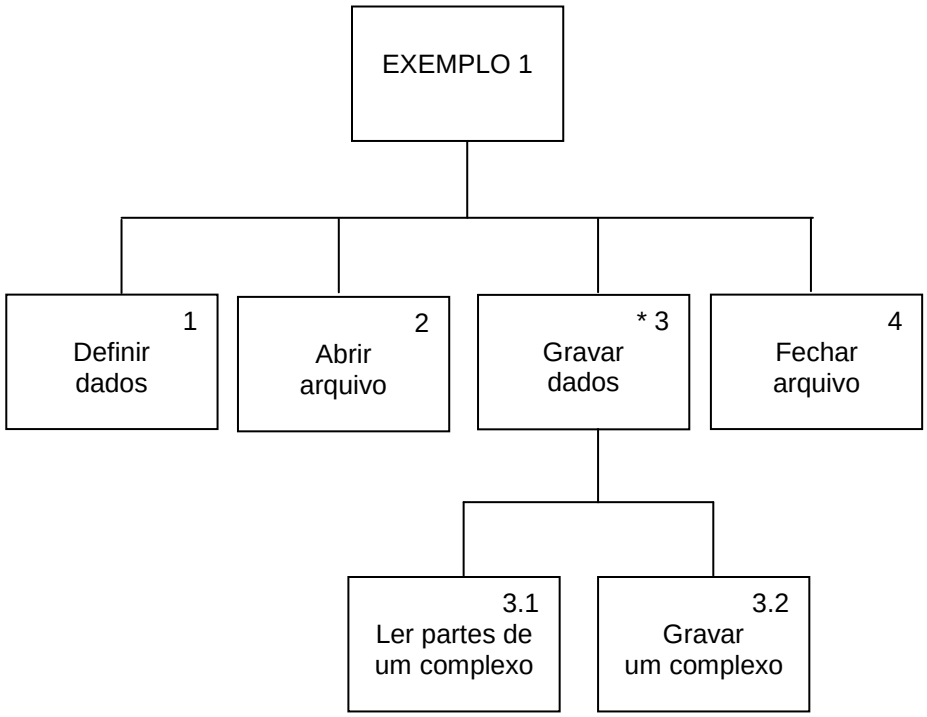
O posicionamento será possível se o arquivo já tiver sido definido e aberto.
A posição deverá ser válida. A primeira posição terá valor igual a zero.

Exemplos.

Exemplo 1.

Fazer um algoritmo para:
- definir um tipo de dados capaz de lidar com um valor complexo, contendo uma parte real e outra imaginária;
- ler dados do teclado para cada uma destas partes e gravá-los em um arquivo.

Diagrama funcional:



Análise de dados:

- Dados do problema:

Dado	Tipo	Valor inicial	Função
X	(real, real)	-	armazenar a parte real e a parte imaginária
A	arquivo	-	texto com complexos
Y	inteiro	-	auxiliar para a repetição

- Avaliação da solução:

Parte real	Parte imaginária	Complexo
0	1	(0, 1)
1	0	(1, 0)
1	1	(1, 1)
3	4	(3, 4)

Algoritmo:

Esboço:

Primeira versão, só comentários.

Exemplo 1	v.1
Ação	Bloco
! definir dados	1
! abrir arquivo para gravação	2
! gravar dados	3
! ler partes de um complexo	3.1
! gravar um complexo	3.2
! fechar arquivo	4

Segunda versão, refinar o primeiro bloco.

Exemplo 1	v.2
Ação	Bloco
! definir dados ! definir tipos tipo COMPLEXO = registro REAL Re ! parte real REAL Im ! parte imaginária fim registro ! COMPLEXO tipo ARQCOMP = arquivo de COMPLEXO	1
! abrir arquivo para gravação	2
! gravar dados	3
! ler partes de um complexo	3.1
! gravar um complexo	3.2
! fechar arquivo	4

Terceira versão, continuar o refinamento do primeiro bloco.

Exemplo 1	v.3
Ação	Bloco
! definir dados ! definir tipos tipo COMPLEXO = registro REAL Re ! parte real REAL Im ! parte imaginária fim registro ! COMPLEXO tipo ARQCOMP = arquivo de COMPLEXO ! definir dados dos tipos acima COMPLEXO X ARQCOMP A inteiro Y	1
! abrir arquivo para gravação	2
! gravar dados	3
! ler partes de um complexo	3.1
! gravar um complexo	3.2
! fechar arquivo	4

Quarta versão, refinar o segundo e o quarto blocos.

Exemplo 1	v.4
Ação	Bloco

! definir dados
  ! definir tipos
  tipo COMPLEXO
  = registro
  REAL Re ! parte real
  REAL Im ! parte imaginária
  fim registro ! COMPLEXO
  tipo ARQCOMP = arquivo de COMPLEXO
  ! definir dados dos tipos acima
  COMPLEXO X
  ARQCOMP A
  inteiro Y
	1

! abrir arquivo para gravação abrir (ARQCOMP, "COMPLEXOS.TXT", gravação)	2
! gravar dados	3
! ler partes de um complexo	3.1
! gravar um complexo	3.2
! fechar arquivo fechar (ARQCOMP)	4

Quinta versão, refinar o terceiro bloco.

Exemplo 1	v.5
Ação	Bloco

! definir dados
  ! definir tipos
  tipo COMPLEXO
  = registro
  REAL Re ! parte real
  REAL Im ! parte imaginária
  fim registro ! COMPLEXO
  tipo ARQCOMP = arquivo de COMPLEXO
  ! definir dados dos tipos acima
  COMPLEXO X
  ARQCOMP A
  inteiro Y
	1

! abrir arquivo para gravação abrir (ARQCOMP, "COMPLEXOS.TXT", gravação)	2
! gravar dados	3
Y = 1	
repetir enquanto (Y ≠ 0)	
! ler partes de um complexo	3.1
! gravar um complexo	3.2
! verificar se ha' mais dados	
Y ← teclado	
! fechar arquivo fechar (ARQCOMP)	4

Sexta versão, continuar o refinamento do terceiro bloco.

Exemplo 1	v.6
Ação	Bloco

! definir dados
  ! definir tipos
  tipo COMPLEXO
  = registro
  REAL Re ! parte real
  REAL Im ! parte imaginária
  fim registro ! COMPLEXO
  tipo ARQCOMP = arquivo de COMPLEXO
  ! definir dados dos tipos acima
  COMPLEXO X
  ARQCOMP A
  inteiro Y
	1

! abrir arquivo para gravação abrir (ARQCOMP, "COMPLEXOS.TXT", gravação)	2
! gravar dados	3
Y = 1	
repetir enquanto (Y ≠ 0)	
! ler partes de um complexo (X.Re, X.Im) ← teclado	3.1
! gravar um complexo ARQCOMP ← X	3.2
! verificar se ha' mais dados	
Y ← teclado	
! fechar arquivo fechar (ARQCOMP)	4

Programa em SCILAB:

```
% Exemplo 1.
% Gravar complexos em arquivo texto.
%
% 1. definir dados
   Y = 0;                          % auxiliar para a repeticao
%
% 2. abrir arquivo para gravacao
  ARQCOMP = fopen ( 'COMPLEXOS.TXT', 'w' );
%
% 3. ler dados
   Y = 1;
   while ( Y ~= 0 )
   % 3.1 ler as partes de um complexo
      X.Re = input ( ' \nQual a parte real ? ' );
      X.Im = input  ( ' \nQual a parte imaginaria ? ' );
   % 3.2 gravar um complexo
      fprintf ( ARQCOMP, '%f %f\n', X.Re, X.Im );
   % 3.3 verificar se ha' mais dados
      Y = input ( ' \nMais dados (Sim=1, Nao=0) ? ' );
   end % fim while
 %
 % 4. fechar arquivo
    fclose ( ARQCOMP );

% pausa para terminar
   printf  ( ' \n\nPressionar qualquer tecla para terminar.' );
   pause;
% fim do programa
```

Programa em C++:

```
// Exemplo 1a.
// Gravar complexos em arquivo texto.
//
// bibliotecas necessarias
#include <fstream>            // para arquivos
#include <iostream>
using namespace std;
//
// definir tipos globais
//
  typedef struct
  {                            // registro
    float Re;                  //   parte real
    float Im;                  //   parte imaginaria
  } COMPLEXO;                  // fim registro
//
// parte principal
//
int main (void)
{
 // 1. definir dados
    COMPLEXO X;
    fstream        ARQCOMP;
    int            Y;

 // 2. abrir arquivo para gravacao
    ARQCOMP.open ( "COMPLEXOS.TXT", ios::out );

 // 3. ler dados
   Y = 1;
   while ( Y != 0 )
   { // 3.1 ler partes de um complexo
       cout << "\nQual a parte real ? ";
       cin   >> X.Re;
       cout << "\nQual a parte imaginaria ? ";
       cin   >> X.Im;
     // 3.2 gravar um complexo
       ARQCOMP << X.Re << " " << X.Im << "\n";
     // 3.3 verificar se ha' mais dados
      cout << "\nMais dados (Sim=1, Nao=0) ? ";
      cin   >> Y;
   } // fim while

 // 4. fechar arquivo
    ARQCOMP.close ( );

// pausa para terminar
   cout << "\nPressionar qualquer tecla para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Outra versão em C++:

```
// Exemplo 1b.
// Gravar complexos em arquivo binario.
//
// bibliotecas necessarias
#include <fstream>            // para arquivos
#include <iostream>
using namespace std;
//
// definir tipos globais
//
  typedef struct
  {                            // registro
    float Re;                  //   parte real
    float Im;                  //   parte imaginaria
  } COMPLEXO;                  // fim registro
//
// parte principal
//
int main (void)
{
 // 1. definir dados
    COMPLEXO X;
    fstream        ARQCOMP;
    int            Y;

 // 2. abrir arquivo para gravacao
    ARQCOMP.open ( "COMPLEXOS.TXT", ios::binary | ios::out );

 // 3. ler dados
   Y = 1;
   while ( Y != 0 )
   { // 3.1 ler partes de um complexo
       cout << "\nQual a parte real ? ";
       cin   >> X.Re;
       cout << "\nQual a parte imaginaria ? ";
       cin   >> X.Im;
     // 3.2 gravar um complexo
       ARQCOMP.write ( (char *) &X, sizeof ( COMPLEXO) );
     // 3.3 verificar se ha' mais dados
      cout << "\nMais dados (Sim=1, Nao=0) ? ";
      cin   >> Y;
   } // fim while

 // 4. fechar arquivo
    ARQCOMP.close ( );

// pausa para terminar
   cout << "\nPressionar qualquer tecla para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Programa em C#:

```
/*
 * Exemplo 1
 * Gravar complexos em um arquivo.
 */
using System;
using System.IO;

class COMPLEXO
{
  public double Re = 0.0;            // parte real
  public double Im = 0.0;            // parte imaginaria
} // fim COMPLEXO class

class Exemplo_1
{
   public static void Main ( )
   {
   // 1. definir dados
      COMPLEXO X = new COMPLEXO ( );
      int Y;

   // 2. abrir arquivo para gravacao
     TextWriter ARQCOMP = new StreamWriter ( "COMPLEXOS.TXT" );

   // 3. ler dados
      Y = 1;
      while ( Y != 0 )
      { // 3.1 ler partes de um complexo
           Console.Write ( "\nQual a parte real ? " );
           X.Re = double.Parse     ( Console.ReadLine ( ) );
           Console.Write ("\nQual a parte imaginaria ? " );
           X.Im = double.Parse     ( Console.ReadLine ( ) );
        // 3.2 gravar um complexo
           ARQCOMP.WriteLine ( X.Re );
           ARQCOMP.WriteLine ( X.Im );
        // 3.3 verificar se ha' mais dados
           Console.WriteLine ( "\nMais dados (Sim=1,Nao=0) ? " );
           Y = int.Parse ( Console.ReadLine ( ) );
      } // fim while

   // 4. fechar arquivo
      ARQCOMP.Close ( );
   // pausa para terminar
      Console.Write       ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_1 class
```

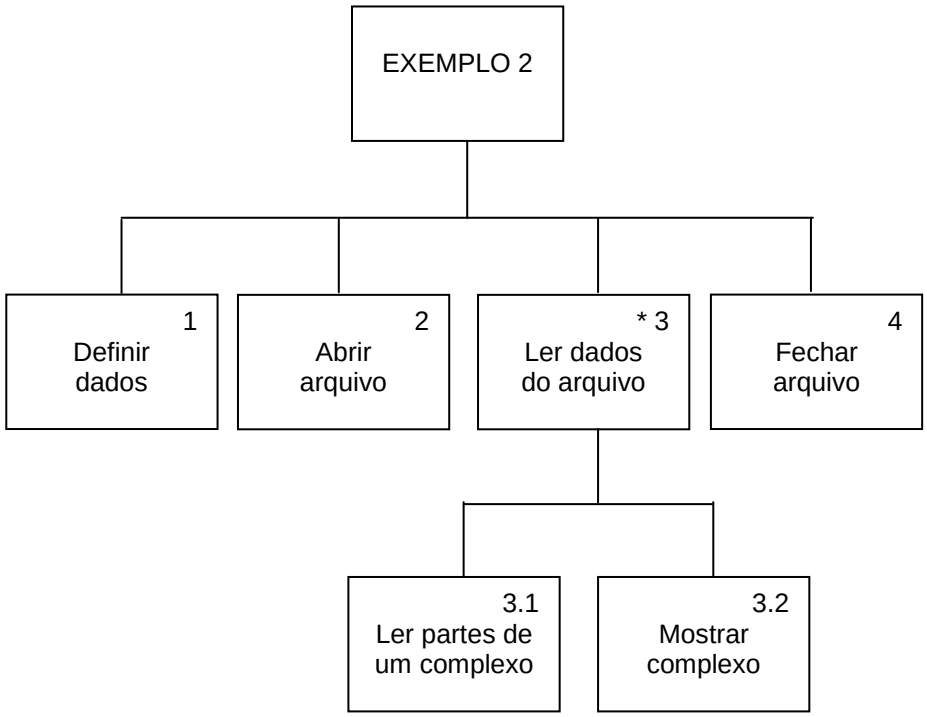
Programa em Java:

```
/**
  * Exemplo 1
  * Gravar complexos em um arquivo.
  */

// ------------------------------------------------ classes necessarias
import IO.*;                                  // IO.jar   deve ser acessivel
// ------------------------------------------------ definicao de classe

class COMPLEXO
{
  public double Re = 0.0;            // parte real
  public double Im = 0.0;            // parte imaginaria
} // fim COMPLEXO class

class Exemplo_1
{
   public static void main ( String [ ] args )
   {
   // 1. definir dados
      COMPLEXO X = new COMPLEXO ( );
      int Y;

   // 2. abrir arquivo para gravacao
      FILE ARQCOMP = new FILE ( FILE.OUTPUT, "COMPLEXOS.TXT" );

   // 3. ler dados
      Y = 1;
      while ( Y != 0 )
      { // 3.1 ler partes de um complexo
           X.Re = IO.readdouble ( "\nQual a parte real ? " );
           X.Im = IO.readdouble ( "\nQual a parte imaginaria ? " );
        // 3.2 gravar um complexo
           ARQCOMP.println ( "" + X.Re );
           ARQCOMP.println ( "" + X.Im );
        // 3.3 verificar se ha' mais dados
           Y = IO.readint ( "\nMais dados (Sim=1, Nao=0) ? " );
      } // fim while

   // 4. fechar arquivo
      ARQCOMP.close ( );
   // pausa para terminar
      IO.pause ( "\nPressionar ENTER para terminar." );
   } // end main ( )

} // fim Exemplo_1 class
```

Exemplo 2.

Fazer um algoritmo para:
- ler valores complexos gravados em um arquivo do tipo texto, um par de valores por linha;
- mostrar na tela os valores lidos.

Diagrama funcional:



Análise de dados:

- Dados do problema:

Dado	Tipo	Valor inicial	Função
X	(real, real)	-	armazenar a parte real e a parte imaginária
A	arquivo	-	texto com complexos
Y	inteiro	-	auxiliar para a repetição

Algoritmo:

Esboço:

Primeira versão, só comentários.

Exemplo 2	v.1
Ação	Bloco
! definir dados	1
! abrir arquivo para leitura	2
! ler dados de arquivo	3
! ler partes de um complexo	3.1
! mostrar um complexo	3.2
! fechar arquivo	4

Segunda versão, continuar o refinamento do primeiro bloco.

Exemplo 2	v.2
Ação	Bloco
! definir dados ! definir tipos tipo COMPLEXO = registro REAL Re ! parte real REAL Im ! parte imaginária fim registro ! COMPLEXO tipo ARQCOMP = arquivo de COMPLEXO ! definir dados dos tipos acima COMPLEXO X ARQCOMP A inteiro Y	1
! abrir arquivo para leitura	2
! ler dados de arquivo	3
! ler partes de um complexo	3.1
! mostrar um complexo	3.2
! fechar arquivo	4

Terceira versão, refinar o segundo e o quarto blocos.

Exemplo 2	v.3
Ação	Bloco

! definir dados
  ! definir tipos
  tipo COMPLEXO
  = registro
  REAL Re ! parte real
  REAL Im ! parte imaginária
  fim registro ! COMPLEXO
  tipo ARQCOMP = arquivo de COMPLEXO
  ! definir dados dos tipos acima
  COMPLEXO X
  ARQCOMP A
  inteiro Y
	1

! abrir arquivo para leitura abrir (ARQCOMP, "COMPLEXOS.TXT", leitura)	2
! ler dados de arquivo	3
! ler partes de um complexo	3.1
! mostrar um complexo	3.2
! fechar arquivo fechar (ARQCOMP)	4

Quarta versão, refinar o terceiro bloco.

Exemplo 2	v.4
Ação	Bloco

! definir dados
  ! definir tipos
  tipo COMPLEXO
  = registro
  REAL Re ! parte real
  REAL Im ! parte imaginária
  fim registro ! COMPLEXO
  tipo ARQCOMP = arquivo de COMPLEXO
  ! definir dados dos tipos acima
  COMPLEXO X
  ARQCOMP A
  inteiro Y
	1

! abrir arquivo para leitura abrir (ARQCOMP, "COMPLEXOS.TXT", leitura)	2
! ler dados de arquivo	3
! ler partes do primeiro complexo	3.1
repetir enquanto (~ FDA (ARQCOMP))	
! mostrar um complexo	3.2
! verificar se ha' mais dados	3.3
! fechar arquivo fechar (ARQCOMP)	4

Quinta versão, continuar o refinamento do terceiro bloco.

Exemplo 2	v.5
Ação	Bloco

! definir dados
  ! definir tipos
  tipo COMPLEXO
  = registro
  REAL Re ! parte real
  REAL Im ! parte imaginária
  fim registro ! COMPLEXO
  tipo ARQCOMP = arquivo de COMPLEXO
  ! definir dados dos tipos acima
  COMPLEXO X
  ARQCOMP A
  inteiro Y
	1

! abrir arquivo para leitura abrir (ARQCOMP, "COMPLEXOS.TXT", leitura)	2
! ler dados de arquivo	3
! ler partes do primeiro complexo (X.Re, X.Im) ← ARQCOMP	3.1
repetir enquanto (~ FDA (ARQCOMP))	
! mostrar um complexo tela ← ("(", X.Re, ",", X.Im, ")")	3.2
! verificar se ha' mais dados (X.Re, X.Im) ← ARQCOMP	3.3
! fechar arquivo fechar (ARQCOMP)	4

Programa em SCILAB:

```
% Exemplo 2.
% Ler arquivo e mostrar complexos.
%
% 2. abrir arquivo para leitura
  ARQCOMP = fopen ( 'COMPLEXOS.TXT', 'r' );
%
% 3. ler dados
  % 3.1 ler as partes do primeiro complexo
     X.Re = fscanf ( ARQCOMP, '%f', 1 );
     X.Im  = fscanf ( ARQCOMP, '%f', 1 );
    % reperir enquanto nao fim de arquivo
       while ( ! feof ( ARQCOMP ) )
       % 3.2 mostrar um complexo
          printf ( '(%f, %f)\n', X.Re, X.Im );
       % 3.3 verificar se ha' mais dados
          X.Re = fscanf ( ARQCOMP, '%f', 1 );
          X.Im  = fscanf ( ARQCOMP, '%f', 1 );
       end % fim while
%
% 4. fechar arquivo
   fclose ( ARQCOMP );

% pausa para terminar
   printf  ( ' \n\nPressionar qualquer tecla para terminar.' );
   pause;
% fim do programa
```

Programa em C++:

```
// Exemplo 2a.
// Ler arquivo texto e mostrar complexos.
//
// bibliotecas necessarias
#include <fstream>            // para arquivos
#include <iostream>
using namespace std;
//
// definir tipos globais
//
  typedef struct
  {                            // registro
    float Re;                  //   parte real
    float Im;                  //   parte imaginaria
  } COMPLEXO;                  // fim registro
//
// parte principal
//
int main (void)
{
 // 1. definir dados
    COMPLEXO X;
    fstream        ARQCOMP;
    int            Y;

 // 2. abrir arquivo para leitura
    ARQCOMP.open ( "COMPLEXOS.TXT", ios::in );

 // 3. ler dados de arquivo
   // 3.1 ler partes de um complexo
      ARQCOMP >> X.Re >> X.Im;
   // repetir enquanto nao FDA ( ARQCOMP )
      while ( ! ARQCOMP.eof ( ) )
      { // 3.2 mostrar um complexo
          cout << "\n(" << X.Re << "," << X.Im << ")";
        // 3.3 verificar se ha' mais dados
          ARQCOMP >> X.Re >> X.Im;
      } // fim while

 // 4. fechar arquivo
    ARQCOMP.close ( );

// pausa para terminar
   cout << "\nPressionar qualquer tecla para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Programa em C++:

```
// Exemplo 2b.
// Ler arquivo binario e mostrar complexos.
//
// bibliotecas necessarias
#include <fstream>            // para arquivos
#include <iostream>
using namespace std;
//
// definir tipos globais
//
  typedef struct
  {                            // registro
    float Re;                  //   parte real
    float Im;                  //   parte imaginaria
  } COMPLEXO;                  // fim registro
//
// parte principal
//
int main (void)
{
 // 1. definir dados
    COMPLEXO X;
    fstream        ARQCOMP;
    int            Y;

 // 2. abrir arquivo para gravacao
    ARQCOMP.open ( "COMPLEXOS.TXT", ios::binary | ios::in );

 // 3. ler dados de arquivo
   // 3.1 ler partes de um complexo
      ARQCOMP.read ( (char *) &X, sizeof ( COMPLEXO ) );
   // repetir enquanto não FDA ( ARQCOMP )
      while ( ! ARQCOMP.eof ( ) )
      { // 3.2 mostrar um complexo
          cout << "\n(" << X.Re << "," << X.Im << ")";
        // 3.3 verificar se ha' mais dados
          ARQCOMP.read ( (char *) &X, sizeof ( COMPLEXO ) );
      } // fim while

 // 4. fechar arquivo
    ARQCOMP.close ( );

// pausa para terminar
   cout << "\nPressionar qualquer tecla para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Programa em C#:

```
/*
 * Exemplo 2
 * Ler arquivo texto e mostrar complexos.
 */
using System;
using System.IO;

class COMPLEXO
{
  public double Re = 0.0;           // parte real
  public double Im = 0.0;           // parte imaginaria
} // fim COMPLEXO class

class Exemplo_2
{
   public static void Main ( )
   {
   // 1. definir dados
      COMPLEXO X = new COMPLEXO ( );
      string Linha;

   // 2. abrir arquivo para leitura
      TextReader ARQCOMP = new StreamReader ( "COMPLEXOS.TXT" );

   // 3. ler dados de arquivo
      // 3.1 ler partes de um complexo
         Linha = ARQCOMP.ReadLine ( );
      // repetir enquanto nao FDA ( ARQCOMP )
         while ( Linha != null )
         {
             X.Re = double.Parse ( Linha );
             X.Im  = double.Parse ( ARQCOMP.ReadLine ( ) );
           // 3.2 mostrar um complexo
             Console.WriteLine ( "\n(" + X.Re + "," + X.Im + ")" );
           // 3.3 verificar se ha' mais dados
             Linha = ARQCOMP.ReadLine ( );
         } // fim while

   // 4. fechar arquivo
      ARQCOMP.Close ( );
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_2 class
```

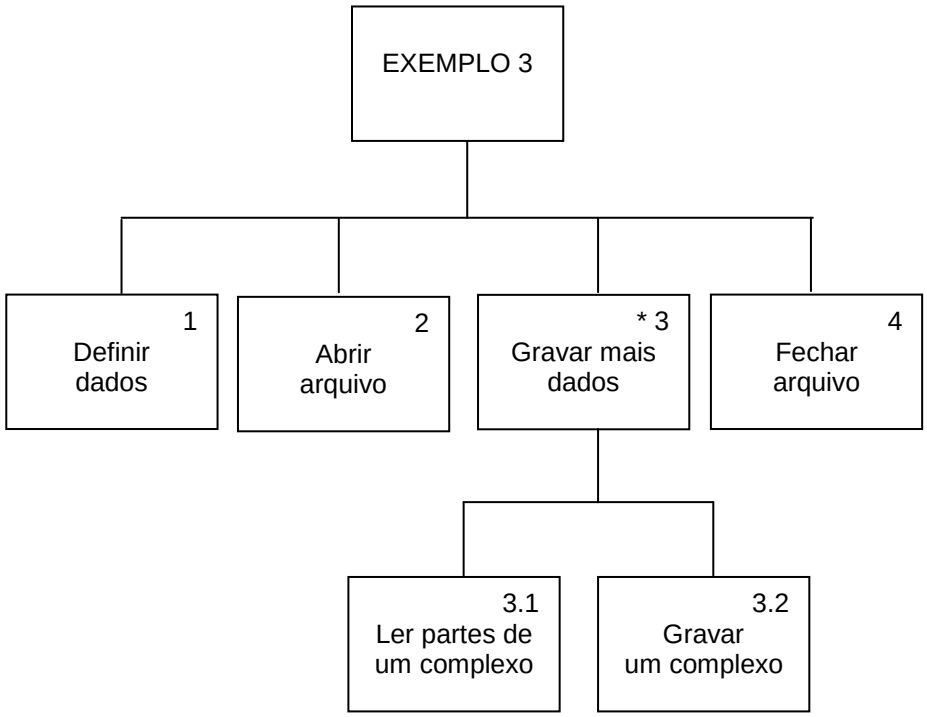
Programa em Java:

```
/**
  * Exemplo 2
  * Ler arquivo texto e mostrar complexos.
  */

// ---------------------------------------------- classes necessarias
import IO.*;                          // IO.jar   deve ser acessivel
// ---------------------------------------------- definicao de classe

class COMPLEXO
{
  public double Re = 0.0;             // parte real
  public double Im = 0.0;             // parte imaginaria
} // fim COMPLEXO class

class Exemplo_2
{
   public static void main ( String [ ] args )
   {
   // 1. definir dados
      COMPLEXO X = new COMPLEXO ( );
      int      Y;
      String Linha = "";

   // 2. abrir arquivo para leitura
      FILE ARQCOMP = new FILE ( FILE.INPUT, "COMPLEXOS.TXT" );

   // 3. ler dados de arquivo
      // 3.1 ler partes de um complexo
         Linha = ARQCOMP.read ( );
      // repetir enquanto nao FDA ( ARQCOMP )
         while ( ! ARQCOMP.eof ( ) )
         {
             X.Re  = Double.parseDouble ( Linha );
             Linha = ARQCOMP.read ( );
             X.Im  = Double.parseDouble ( Linha );
           // 3.2 mostrar um complexo
             IO.println ( "\n(" + X.Re + "," + X.Im + ")" );
           // 3.3 verificar se ha' mais dados
             Linha = ARQCOMP.read ( );
         } // fim while

   // 4. fechar arquivo
      ARQCOMP.close ( );
   // pausa para terminar
      IO.pause ( "\nPressionar ENTER para terminar." );
   } // end main ( )

} // fim Exemplo_2 class
```

Exemplo 3.

Fazer um algoritmo para:
- acrescentar valores complexos a um arquivo;
- mostrar na tela os valores lidos.

Diagrama funcional:



Análise de dados:

- Dados do problema:

Dado	Tipo	Valor inicial	Função
X	(real, real)	-	armazenar a parte real e a parte imaginária
A	arquivo	-	texto com complexos
Y	inteiro	-	auxiliar para a repetição

- Avaliação da solução:

Parte real	Parte imaginária	Complexo
0	1	(0, 1)
1	0	(1, 0)
1	1	(1, 1)
3	4	(3, 4)

Algoritmo:

Esboço:

Primeira versão, só comentários.

Exemplo 3	v.1
Ação	Bloco
! definir dados	1
! abrir arquivo para acréscimo	2
! gravar mais dados	3
! ler partes de um complexo	3.1
! gravar um complexo	3.2
! fechar arquivo	4

Segunda versão, refinar o primeiro bloco.

Exemplo 3	v.2
Ação	Bloco
! definir dados ! definir tipos tipo COMPLEXO = registro REAL Re ! parte real REAL Im ! parte imaginária fim registro ! COMPLEXO tipo ARQCOMP = arquivo de COMPLEXO ! definir dados dos tipos acima COMPLEXO X ARQCOMP A inteiro Y	1
! abrir arquivo para acréscimo	2
! gravar mais dados	3
! ler partes de um complexo	3.1
! gravar um complexo	3.2
! fechar arquivo	4

Terceira versão, refinar o segundo e o quarto blocos.

Exemplo 3	v.3
Ação	Bloco

! definir dados
  ! definir tipos
  tipo COMPLEXO
  = registro
  REAL Re ! parte real
  REAL Im ! parte imaginária
  fim registro ! COMPLEXO
  tipo ARQCOMP = arquivo de COMPLEXO
  ! definir dados dos tipos acima
  COMPLEXO X
  ARQCOMP A
  inteiro Y
	1

! abrir arquivo para acréscimo abrir (ARQCOMP, "COMPLEXOS.TXT", acréscimo)	2
! gravar mais dados	3
! ler partes de um complexo	3.1
! gravar um complexo	3.2
! fechar arquivo fechar (ARQCOMP)	4

Quarta versão, refinar o terceiro bloco.

Exemplo 3	v.4
Ação	Bloco

! definir dados
  ! definir tipos
  tipo COMPLEXO
  = registro
  REAL Re ! parte real
  REAL Im ! parte imaginária
  fim registro ! COMPLEXO
  tipo ARQCOMP = arquivo de COMPLEXO
  ! definir dados dos tipos acima
  COMPLEXO X
  ARQCOMP A
  inteiro Y
	1

! abrir arquivo para acréscimo abrir (ARQCOMP, "COMPLEXOS.TXT", acréscimo)	2
! gravar mais dados	3
Y = 1	
repetir enquanto (Y ≠ 0)	
! ler partes de um complexo	3.1
! gravar um complexo	3.2
! verificar se ha' mais dados	
Y ← teclado	
! fechar arquivo fechar (ARQCOMP)	4

Quinta versão, continuar o refinamento do terceiro bloco.

Exemplo 1	v.5
Ação	Bloco

! definir dados
  ! definir tipos
  tipo COMPLEXO
  = registro
  REAL Re ! parte real
  REAL Im ! parte imaginária
  fim registro ! COMPLEXO
  tipo ARQCOMP = arquivo de COMPLEXO
  ! definir dados dos tipos acima
  COMPLEXO X
  ARQCOMP A
  inteiro Y
	1

! abrir arquivo para acréscimo abrir (ARQCOMP, "COMPLEXOS.TXT", acréscimo)	2
! gravar mais dados	3
Y = 1	
repetir enquanto (Y ≠ 0)	
! ler partes de um complexo (X.Re, X.Im) ← teclado	3.1
! gravar um complexo ARQCOMP ← X	3.2
! verificar se ha' mais dados	
Y ← teclado	
! fechar arquivo fechar (ARQCOMP)	4

Programa em SCILAB:

```
% Exemplo 3.
% Gravar mais complexos em arquivo texto.
%
% 1. definir dados
   Y = 0;                          % auxiliar para a repeticao
%
% 2. abrir arquivo para acrescimo
  ARQCOMP = fopen ( 'COMPLEXOS.TXT', 'a' );
%
% 3. ler dados
   Y = 1;
   while ( Y ~= 0 )
   % 3.1 ler as partes de um complexo
      X.Re = input ( ' \nQual a parte real ? ' );
      X.Im = input  ( ' \nQual a parte imaginaria ? ' );
   % 3.2 gravar um complexo
      fprintf ( ARQCOMP, '%f %f\n', X.Re, X.Im );
   % 3.3 verificar se ha' mais dados
      Y = input ( ' \nMais dados (Sim=1, Nao=0) ? ' );
   end % fim while
 %
 % 4. fechar arquivo
    fclose ( ARQCOMP );

% pausa para terminar
   printf  ( ' \n\nPressionar qualquer tecla para terminar.' );
   pause;
% fim do programa
```

Programa em C++:

```
// Exemplo 3a.
// Gravar mais complexos em arquivo texto.
//
// bibliotecas necessarias
#include <fstream>            // para arquivos
#include <iostream>
using namespace std;
//
// definir tipos globais
//
  typedef struct
  {                            // registro
    float Re;                  //   parte real
    float Im;                  //   parte imaginaria
  } COMPLEXO;                  // fim registro
//
// parte principal
//
int main (void)
{
 // 1. definir dados
    COMPLEXO X;
    fstream        ARQCOMP;
    int            Y;
 // 2. abrir arquivo para acrescimo
    ARQCOMP.open ( "COMPLEXOS.TXT", ios::app );

 // 3. ler dados
   Y = 1;
   while ( Y != 0 )
   { // 3.1 ler partes de um complexo
        cout << "\nQual a parte real ? ";
        cin   >> X.Re;
        cout << "\nQual a parte imaginaria ? ";
        cin   >> X.Im;
     // 3.2 gravar um complexo
        ARQCOMP << X.Re << " " << X.Im << "\n";
     // 3.3 verificar se há mais dados
       cout << "\nMais dados (Sim=1, Nao=0) ? ";
       cin   >> Y;
   } // fim while

 // 4. fechar arquivo
    ARQCOMP.close ( );

// pausa para terminar
   cout << "\nPressionar qualquer tecla para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Outra versão em C++:

```
// Exemplo 3b.
// Gravar mais complexos em arquivo binario.
//
// bibliotecas necessarias
#include <fstream>            // para arquivos
#include <iostream>
using namespace std;
//
// definir tipos globais
//
  typedef struct
  {                            // registro
    float Re;                  //   parte real
    float Im;                  //   parte imaginaria
  } COMPLEXO;                  // fim registro
//
// parte principal
//
int main (void)
{
 // 1. definir dados
    COMPLEXO X;
    fstream        ARQCOMP;
    int            Y;
 // 2. abrir arquivo para acrescimo
    ARQCOMP.open ( "COMPLEXOS.TXT", ios::binary | ios::app );
 // 3. ler dados
   Y = 1;
   while ( Y != 0 )
   { // 3.1 ler partes de um complexo
        cout << "\nQual a parte real ? ";
        cin   >> X.Re;
        cout << "\nQual a parte imaginaria ? ";
        cin   >> X.Im;
     // 3.2 gravar um complexo
        ARQCOMP.write ( (char *) &X, sizeof ( COMPLEXO ) );
     // 3.3 verificar se há mais dados
       cout << "\nMais dados (Sim=1, Nao=0) ? ";
       cin   >> Y;
   } // fim while
 // 4. fechar arquivo
    ARQCOMP.close ( );

// pausa para terminar
   cout << "\nPressionar qualquer tecla para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Programa em C#:

```
/*
 * Exemplo 3
 * Gravar mais complexos em arquivo texto.
 */
using System;
using System.IO;

class COMPLEXO
{
  public double Re = 0.0;           // parte real
  public double Im = 0.0;           // parte imaginaria
} // fim COMPLEXO class

class Exemplo_3
{
   public static void Main ( )
   {
   // 1. definir dados
      COMPLEXO X = new COMPLEXO ( );
      int Y;
   // 2. abrir arquivo para leitura
        FileStream ARQUIVO = new FileStream ( "COMPLEXOS.TXT",
                                                FileMode.Append );
        StreamWriter ARQCOMP = new StreamWriter ( ARQUIVO );
   // 3. ler dados
      Y = 1;
      while ( Y != 0 )
      { // 3.1 ler partes de um complexo
           Console.Write ( "\nQual a parte real ? " );
           X.Re = double.Parse   ( Console.ReadLine ( ) );
           Console.Write ("\nQual a parte imaginaria ? " );
           X.Im = double.Parse   ( Console.ReadLine ( ) );
        // 3.2 gravar um complexo
           ARQCOMP.WriteLine ( X.Re );
           ARQCOMP.WriteLine ( X.Im );
        // 3.3 verificar se ha' mais dados
           Console.WriteLine ( "\nMais dados (Sim=1,Nao=0) ? " );
           Y = int.Parse ( Console.ReadLine ( ) );
      } // fim while
   // 4. fechar arquivo
      ARQCOMP.Close ( );
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_3 class
```

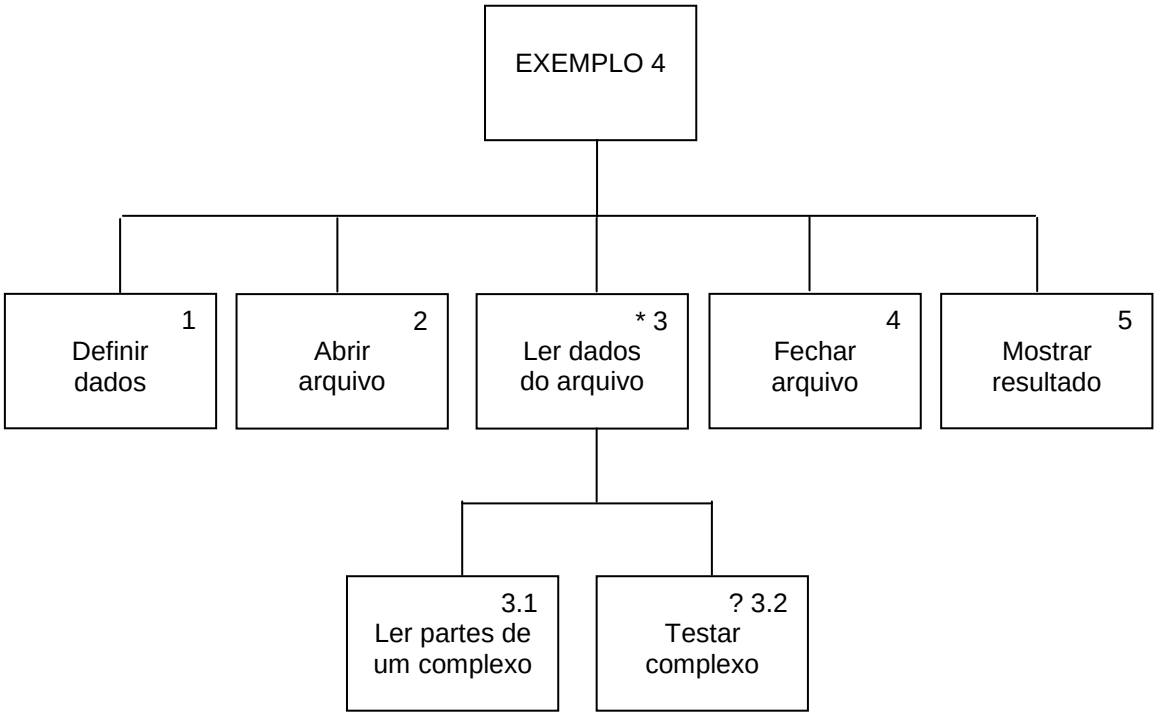
Programa em Java:

```
/**
  * Exemplo 3
  * Gravar mais complexos em arquivo texto.
  */

// ------------------------------------------------ classes necessarias
import IO.*;                            // IO.jar   deve ser acessivel
// ------------------------------------------------ definicao de classe

class COMPLEXO
{
  public double Re = 0.0;           // parte real
  public double Im = 0.0;           // parte imaginaria
} // fim COMPLEXO class

class Exemplo_3
{
   public static void main ( String [ ] args )
   {
   // 1. definir dados
      COMPLEXO X = new COMPLEXO ( );
      int Y;
   // 2. abrir arquivo para leitura
      FILE ARQCOMP = new FILE ( FILE.APPEND, "COMPLEXOS.TXT" );
   // 3. ler dados
      Y = 1;
      while ( Y != 0 )
      { // 3.1 ler partes de um complexo
           X.Re = IO.readdouble ( "\nQual a parte real ? " );
           X.Im  = IO.readdouble ( "\nQual a parte imaginaria ? " );
        // 3.2 gravar um complexo
           ARQCOMP.println ( "" + X.Re + " " + X.Im );
        // 3.3 verificar se há mais dados
           Y = IO.readint ( "\nMais dados (Sim=1, Nao=0) ? " );
      } // fim while
   // 4. fechar arquivo
      ARQCOMP.close ( );
   // pausa para terminar
      IO.pause ( "\nPressionar ENTER para terminar." );
   } // end main ( )

} // fim Exemplo_3 class
```

Exemplo 4.

Fazer um algoritmo para:
- procurar valor complexo gravado em um arquivo do tipo texto, um par de valores por linha;
- mostrar na tela se o valor foi encontrado ou não.

Diagrama funcional:



Análise de dados:

- Dados do problema:

Dado	Tipo	Valor inicial	Função
X	(real, real)	-	armazenar a parte real e a parte imaginária
A	arquivo	-	texto com complexos
Y	inteiro	-	auxiliar para a repetição
PROCURADO	(real, real)	-	armazenar a parte real e a parte imaginária

Algoritmo:

Esboço:

Primeira versão, só comentários.

Exemplo 4	v.1
Ação	Bloco
! definir dados	1
! ler dado a ser procurado	
! abrir arquivo para leitura	2
! ler dados de arquivo	3
! ler partes de um complexo	3.1
! testar um complexo	3.2
! fechar arquivo	4
! mostrar resposta	5

Segunda versão, continuar o refinamento do primeiro bloco.

Exemplo 4	v.2
Ação	Bloco
! definir dados ! definir tipos tipo COMPLEXO = registro REAL Re ! parte real REAL Im ! parte imaginária fim registro ! COMPLEXO tipo ARQCOMP = arquivo de COMPLEXO ! definir dados dos tipos acima COMPLEXO X, PROCURADO ARQCOMP A inteiro Y	1
! ler dado a ser procurado	
! abrir arquivo para leitura	2
! ler dados de arquivo	3
! ler partes de um complexo	3.1
! testar um complexo	3.2
! fechar arquivo	4
! mostrar resposta	5

Terceira versão, refinar o segundo e o quarto blocos.

Exemplo 4	v.3
Ação	Bloco

! definir dados
  ! definir tipos
  tipo COMPLEXO
  = registro
  REAL Re ! parte real
  REAL Im ! parte imaginária
  fim registro ! COMPLEXO
  tipo ARQCOMP = arquivo de COMPLEXO
  ! definir dados dos tipos acima
  COMPLEXO X, PROCURADO
  ARQCOMP A
  inteiro Y
	1

! ler dado a ser procurado (PROCURADO.Re, PROCURADO.Im) ← teclado	
! abrir arquivo para leitura abrir (ARQCOMP, "COMPLEXOS.TXT", leitura)	2
! ler dados de arquivo	3
! ler partes de um complexo	3.1
! testar um complexo	3.2
! fechar arquivo fechar (ARQCOMP)	4
! mostrar resposta	5

Quarta versão, refinar o terceiro bloco.

Exemplo 4	v.4
Ação	Bloco

! definir dados
  ! definir tipos
  tipo COMPLEXO
  = registro
  REAL Re ! parte real
  REAL Im ! parte imaginária
  fim registro ! COMPLEXO
  tipo ARQCOMP = arquivo de COMPLEXO
  ! definir dados dos tipos acima
  COMPLEXO X, PROCURADO
  ARQCOMP A
  inteiro Y
	1

! ler dado a ser procurado (PROCURADO.Re, PROCURADO.Im) ← teclado	
! abrir arquivo para leitura abrir (ARQCOMP, "COMPLEXOS.TXT", leitura)	2
! ler dados de arquivo	3
! ler partes do primeiro complexo	3.1
repetir enquanto (~ FDA (ARQCOMP))	
! testar um complexo	3.2
! verificar se ha' mais dados	3.3
! fechar arquivo fechar (ARQCOMP)	4
! mostrar resposta	5

Quinta versão, continuar o refinamento do terceiro bloco.

Exemplo 4			v.5
Ação			Bloco

! definir dados
  ! definir tipos
  tipo COMPLEXO
  = registro
  REAL Re ! parte real
  REAL Im ! parte imaginária
  fim registro ! COMPLEXO
  tipo ARQCOMP = arquivo de COMPLEXO
  ! definir dados dos tipos acima
  COMPLEXO X, PROCURADO
  ARQCOMP A
  Inteiro Y,
  ENCONTRADO = 0; // resposta
			1

! ler dado a ser procurado (PROCURADO.Re, PROCURADO.Im) ← teclado			
! abrir arquivo para leitura abrir (ARQCOMP, "COMPLEXOS.TXT", leitura)			2
! ler dados de arquivo			3
! ler partes do primeiro complexo (X.Re, X.Im) ← ARQCOMP			3.1
repetir enquanto (~ FDA (ARQCOMP) & ENCONTRADO = 0)			
! testar um complexo			3.2
PROCURADO.Re = X.Re & PROCURADO.Im = X.Im ?	V	ENCONTRADO = 1	
	F	! verificar se ha' mais dados (X.Re, X.Im) ← ARQCOMP	
! fechar arquivo fechar (ARQCOMP)			4
! mostrar resposta			5
ENCONTRADO == 0 ?	V	tela ← "\nValor encontrado."	
	F	tela ← "\nValor nao encontrado."	

Programa em SCILAB:

```
% Exemplo 4.
% Ler arquivo e procurar complexo.
%
% 1. definir dados
   PROCURADO.Re = 0.0;
   PROCURADO.Im  = 0.0;
   ENCONTRADO    = 0;
% ler dado a ser procurado
  fprintf ( '\nQual o numero a ser procurado:\n' );
  PROCURADO.Re = input ( '\nParte real ? ' );
  PROCURADO.Im  = input ( '\nParte imaginaria ? ' );
% 2. abrir arquivo para leitura
  ARQCOMP = fopen ( 'COMPLEXOS.TXT', 'r' );
%
% 3. ler dados
  % 3.1 ler as partes do primeiro complexo
     X.Re = fscanf ( ARQCOMP, '%f', 1 );
     X.Im  = fscanf ( ARQCOMP, '%f', 1 );
    % reperir enquanto nao fim de arquivo
       while ( ! feof ( ARQCOMP ) & ENCONTRADO == 0 )
       % 3.2 testar um complexo
         if ( PROCURADO.Re == X.Re &
              PROCURADO.Im ==  X.Im )
              ENCONTRADO = 1;
         else
       % 3.3 verificar se ha' mais dados
              X.Re = fscanf ( ARQCOMP, '%f', 1 );
              X.Im = fscanf ( ARQCOMP, '%f',  1 );
         end % fim if
       end % fim while
%
% 4. fechar arquivo
   fclose ( ARQCOMP );
%
% 5. mostrar resposta
   if ( ENCONTRADO == 1 )
     printf ( '\nValor encontrado' );
   else
     printf ( '\nValor nao encontrado' );
   end % fim if
% pausa para terminar
   printf  ( ' \n\nPressionar qualquer tecla para terminar.' );
   pause;
% fim do programa
```

Programa em C++:

```
// Exemplo 4a.
// Ler arquivo texto e procurar complexo.
//
// bibliotecas necessarias
#include <fstream>            // para arquivos
#include <iostream>
using namespace std;
//
// definir tipos globais
//
  typedef struct
  {                            // registro
    float Re;                  //   parte real
    float Im;                  //   parte imaginaria
  } COMPLEXO;                  // fim registro
//
// parte principal
//
int main (void)
{
// 1. definir dados
   COMPLEXO X, PROCURADO;
   fstream        ARQCOMP;
   int            Y,
                  ENCONTRADO = 0;
//  ler dado a ser procurado
   cin >> PROCURADO.Re >> PROCURADO.Im;
// 2. abrir arquivo para leitura
      ARQCOMP.open ( "COMPLEXOS.TXT", ios::in );
// 3. ler dados de arquivo
  // 3.1 ler partes de um complexo
     ARQCOMP >> X.Re >> X.Im;
  // repetir enquanto não FDA ( ARQCOMP )
     while ( ! ARQCOMP.eof ( ) && ENCONTRADO == 0 )
     { // 3.2 testar um complexo
         if ( PROCURADO.Re == X.Re &&
              PROCURADO.Im == X.Im )
           ENCONTRADO = 1;
         else
       // 3.3 verificar se há mais dados
           ARQCOMP >> X.Re >> X.Im;
     } // fim while
// 4. fechar arquivo
   ARQCOMP.close ( );
// 5. mostrar o resultado
   if ( ENCONTRADO )
      cout << "\nValor encontrado.";
   else
      cout << "\nValor nao encontrado.";
// pausa para terminar
  cout << "\nPressionar qualquer tecla para terminar.";
  getchar ( );
  return EXIT_SUCCESS;
} // fim do programa
```

Programa em C++:

```
// Exemplo 4b.
// Ler arquivo binario e procurar complexo.
//
// bibliotecas necessarias
#include <fstream>            // para arquivos
#include <iostream>
using namespace std;
//
// definir tipos globais
//
  typedef struct
  {                            // registro
    float Re;                  //   parte real
    float Im;                  //   parte imaginaria
  } COMPLEXO;                  // fim registro
//
// parte principal
//
int main (void)
{
// 1. definir dados
  COMPLEXO X;
  fstream       ARQCOMP;
  int           Y,
                ENCONTRADO = 0;
//  ler dado a ser procurado
   cin >> PROCURADO.Re >> PROCURADO.Im;
// 2. abrir arquivo para gravação
  ARQCOMP.open ( "COMPLEXOS.TXT", ios::binary | ios::in );
// 3. ler dados de arquivo
  // 3.1 ler partes de um complexo
    ARQCOMP.read ( (char *) &X, sizeof ( COMPLEXO ) );
  // repetir enquanto não FDA ( ARQCOMP )
    while ( ! ARQCOMP.eof ( ) && ENCONTRADO == 0)
    { // 3.2 testar um complexo
        if ( PROCURADO.Re == X.Re &&
             PROCURADO.Im == X.Im )
          ENCONTRADO = 1;
        else
       // 3.3 verificar se há mais dados
          ARQCOMP.read ( (char *) &X, sizeof ( COMPLEXO ) );
     } // fim while
// 4. fechar arquivo
  ARQCOMP.close ( );
// 5. mostrar resposta
  if ( ENCONTRADO == 1 )
     cout << "\nValor encontrado.";
  else
      cout << "\nValor nao encontrado.";
// pausa para terminar
  cout << "\nPressionar qualquer tecla para terminar.";
  getchar ( );
  return EXIT_SUCCESS;
} // fim do programa
```

Programa em C#:

```
/*
 * Exemplo 4
 * Ler arquivo texto e procurar complexo.
 */
using System;
using System.IO;

class COMPLEXO
{
  public double Re = 0.0;            // parte real
  public double Im = 0.0;            // parte imaginaria
} // fim COMPLEXO class

class Exemplo_4
{
//
// parte principal
//
   public static void Main ( )
   {
   // 1. definir dados
      COMPLEXO X         = new COMPLEXO ( );
      COMPLEXO PROCURADO  = new COMPLEXO ( );
      string   Linha = " ";
      bool     ENCONTRADO = false;
   // ler valor a ser procurado
      Console.WriteLine ( "\nQual o valor a ser procurado:" );
      Console.Write       ( "\nQual a parte real ? " );
      PROCURADO.Re = double.Parse ( Console.ReadLine ( ) );
      Console.Write ("\nQual a parte imaginaria ? " );
      PROCURADO.Im = double.Parse  ( Console.ReadLine ( ) );
   // 2. abrir arquivo para leitura
      TextReader ARQCOMP = new StreamReader ( "COMPLEXOS.TXT" );
   // 3. ler dados
      // 3.1 ler partes de um complexo
         Linha = ARQCOMP.ReadLine ( );
      // repetir enquanto nao FDA ( ARQCOMP &&  ! ENCONTRADO )
         while ( Linha != null &&  ! ENCONTRADO )
         {
             X.Re = double.Parse ( Linha );
             X.Im  = double.Parse ( ARQCOMP.ReadLine ( ) );
           // 3.2 testar um complexo
             if ( PROCURADO.Re == X.Re &&
                  PROCURADO.Im == X.Im )
                ENCONTRADO = true;
             else
           // 3.3 verificar se ha' mais dados
             Linha = ARQCOMP.ReadLine ( );
         } // fim while
   // 4. fechar arquivo
      ARQCOMP.Close  ( );
```

```
   // 5. mostrar resposta
      if ( ENCONTRADO )
      {
        Console.WriteLine ( "\nValor encontrado."       );
      }
      else
      {
        Console.WriteLine ( "\nValor nao encontrado." );
      } // fim if
   // pausa para terminar
      Console.Write       ( "\nApertar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_4 class
```

Programa em Java:

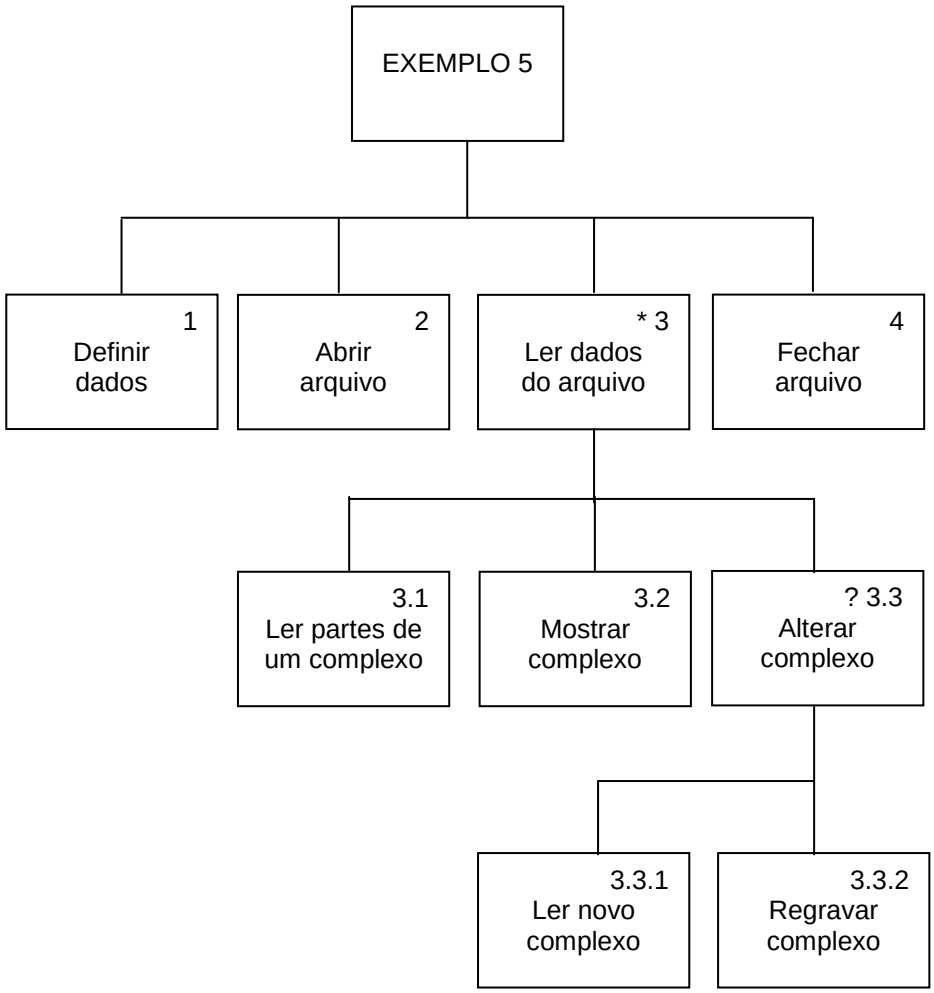
```
/**
  * Exemplo 4
  * Ler arquivo texto e procurar complexo.
  */

// ------------------------------------------------ classes necessarias
import IO.*;                          // IO.jar   deve ser acessivel
// ------------------------------------------------ definicao de classe

class COMPLEXO
{
  public double Re = 0.0;             // parte real
  public double Im = 0.0;             // parte imaginaria
} // fim COMPLEXO class

class Exemplo_4
{
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 1. definir dados
      COMPLEXO X                    = new COMPLEXO ( );
      COMPLEXO PROCURADO  = new COMPLEXO ( );
      int         Linhas,
                  Y, Z;
      String    Linha     = " ";
      boolean ENCONTRADO = false;
   // ler valor a ser procurado
      IO.println ( "\nQual o valor a ser procurado:" );
      PROCURADO.Re = IO.readdouble ( "\nQual a parte real ? " );
      PROCURADO.Im  = IO.readdouble ( "\nQual a parte imaginaria ? " );
   // 2. abrir arquivo para leitura
      FILE ARQCOMP = new FILE ( FILE.INPUT, "COMPLEXOS.TXT" );
   // 3. ler dados
      // 3.1 ler partes de um complexo
         Linha = ARQCOMP.read ( );
      // repetir enquanto nao FDA ( ARQCOMP )
         while ( ! ARQCOMP.eof ( ) &&  ! ENCONTRADO )
         {
            X.Re  = Double.parseDouble ( Linha );
            Linha = ARQCOMP.read ( );
            X.Im  = Double.parseDouble ( Linha );
         // 3.2 testar um complexo
            if ( PROCURADO.Re == X.Re &&
                 PROCURADO.Im == X.Im )
               ENCONTRADO = true;
            else
            // 3.3 verificar se há mais dados
               Linha = ARQCOMP.read ( );
         } // fim while
   // 4. fechar arquivo
      ARQCOMP.close  ( );
```

```
  // 5. mostrar resposta
     if ( ENCONTRADO )
     {
       IO.println ( "\nValor encontrado."       );
     }
     else
     {
       IO.println ( "\nValor nao encontrado." );
     } // fim if
  // pausa para terminar
     IO.pause ( "\nPressionar ENTER para terminar." );
  } // end main ( )

} // fim Exemplo_4 class
```

Exemplo 5.

Fazer um algoritmo para:
- alterar valores complexos gravados em um arquivo;
- mostrar na tela os valores lidos.

Diagrama funcional:



Análise de dados:

- Dados do problema:

Dado	Tipo	Valor inicial	Função
X	(real, real)	-	armazenar a parte real e a parte imaginária
A	arquivo	-	texto com complexos
Y	inteiro	-	auxiliar para a repetição

Algoritmo:

Esboço:

Primeira versão, só comentários.

Exemplo 5	v.1
Ação	Bloco
! definir dados	1
! abrir arquivo para leitura	2
! ler dados de arquivo	3
! ler partes de um complexo	3.1
! mostrar um complexo	3.2
! alterar complexo	3.3
! ler novo complexo	3.3.1
! regravar complexo	3.3.2
! fechar arquivo	4

Segunda versão, continuar o refinamento do primeiro bloco.

Exemplo 5	v.2
Ação	Bloco
! definir dados ! definir tipos tipo COMPLEXO = registro REAL Re ! parte real REAL Im ! parte imaginária fim registro ! COMPLEXO tipo ARQCOMP = arquivo de COMPLEXO ! definir dados dos tipos acima COMPLEXO X ARQCOMP A inteiro Y	1
! abrir arquivo para leitura	2
! ler dados de arquivo	3
! ler partes de um complexo	3.1
! mostrar um complexo	3.2
! alterar complexo	3.3
! ler novo complexo	3.3.1
! regravar complexo	3.3.2
! fechar arquivo	4

Terceira versão, refinar o segundo e o quarto blocos.

Exemplo 5	v.3
Ação	Bloco

! definir dados
  ! definir tipos
  tipo COMPLEXO
  = registro
  REAL Re, Im
  fim registro ! COMPLEXO
  tipo ARQCOMP = arquivo de COMPLEXO
  ! definir dados dos tipos acima
  COMPLEXO X
  ARQCOMP A
  inteiro Y
	1

! abrir arquivo para leitura abrir (ARQCOMP, "COMPLEXOS.TXT", leitura)	2
! ler dados de arquivo	3
! ler partes de um complexo	3.1
! mostrar um complexo	3.2
! alterar complexo	3.3
! ler novo complexo	3.3.1
! regravar complexo	3.3.2
! fechar arquivo fechar (ARQCOMP)	4

Quarta versão, refinar o terceiro bloco.

Exemplo 4	v.4
Ação	Bloco

! definir dados
  ! definir tipos
  tipo COMPLEXO
  = registro
  REAL Re, Im
  fim registro ! COMPLEXO
  tipo ARQCOMP = arquivo de COMPLEXO
  ! definir dados dos tipos acima
  COMPLEXO X
  ARQCOMP A
  inteiro Y
	1

! abrir arquivo para leitura abrir (ARQCOMP, "COMPLEXOS.TXT", leitura)	2
! ler dados de arquivo	3
repetir para (Y = 1 : tamanho("COMPLEXOS.TXT"))	
! ler partes do primeiro complexo X ← ARQCOMP	3.1
! mostrar um complexo tela ← ("(", X.Re, ",", X.Im, ")")	3.2
! alterar complexo	3.3
! ler novo complexo	3.3.1
! regravar complexo	3.3.2
! fechar arquivo fechar (ARQCOMP)	4

Quinta versão, refinar o terceiro bloco.

Exemplo 5	v.5
Ação	Bloco

! definir dados
  ! definir tipos
  tipo COMPLEXO
  = registro
  REAL Re, Im
  fim registro ! COMPLEXO
  tipo ARQCOMP = arquivo de COMPLEXO
  ! definir dados dos tipos acima
  COMPLEXO X
  ARQCOMP A
  inteiro Y, Z
	1

! abrir arquivo para leitura abrir (ARQCOMP, "COMPLEXOS.TXT", leitura)	2
! ler dados de arquivo	3
repetir para (Y = 1 : tamanho("COMPLEXOS.TXT"))	
! ler partes do primeiro complexo X ← ARQCOMP	3.1
! mostrar um complexo tela ← ("(", X.Re, ",", X.Im, ")")	3.2
! alterar complexo	3.3
tela ← "Alterar (Sim=1, Nao=0) ? "	
Z ← teclado	
Z = 1 ? | V | ! ler novo complexo (X.Re, X.Im) ← teclado	3.3.1
Z = 1 ? | V | ! regravar complexo	3.3.2
! fechar arquivo fechar (ARQCOMP)	4

Sexta versão, continuar o refinamento do terceiro bloco.

Exemplo 5	v.6
Ação	Bloco

! definir dados
  ! definir tipos
  tipo COMPLEXO
  = registro
  REAL Re, Im
  fim registro ! COMPLEXO
  tipo ARQCOMP = arquivo de COMPLEXO
  ! definir dados dos tipos acima
  COMPLEXO X
  ARQCOMP A
  inteiro Y, Z
	1

! abrir arquivo para leitura abrir (ARQCOMP, "COMPLEXOS.TXT", leitura)	2
! ler dados de arquivo	3
repetir para (Y = 1 : tamanho("COMPLEXOS.TXT"))	
! ler partes do primeiro complexo X ← ARQCOMP	3.1
! mostrar um complexo tela ← ("(", X.Re, ",", X.Im, ")")	3.2
! alterar complexo	3.3
tela ← "Alterar (Sim=1, Nao=0) ? "	
Z ← teclado	
Z = 1 ? | V | ! ler novo complexo X ← teclado	3.3.1

Z = 1 ? | V | ! regravar complexo
  ! reposicionar na linha Y
  POSICIONAR (ARQCOMP, Y)
  ! gravar na mesma posição
  ARQCOMP ← X
	3.3.2

! fechar arquivo fechar (ARQCOMP)	4

Programa em SCILAB:

```
% Exemplo 5.
% Ler arquivo e alterar complexos.
%
% 1. definir dados
   Linhas = 0;
   P1 = 0;
   P2 = 0;
%
% 2. abrir arquivo para alteracao
   ARQCOMP = fopen ( 'COMPLEXOS.TXT', 'r+' );
%
% 3. ler dados de arquivo
  % determinar o tamanho
      X.Re = fscanf ( ARQCOMP, '%f', 1 );
      X.Im = fscanf ( ARQCOMP, '%f', 1 );
  % repetir enquanto nao FDA ( ARQCOMP )
     while ( ! feof ( ARQCOMP ) )
        Linhas = Linhas + 1;
        X.Re = fscanf ( ARQCOMP, '%f', 1 );
        X.Im = fscanf ( ARQCOMP, '%f', 1 );
     end % fim while
  % reiniciar o arquivo
     STATUS = fseek ( ARQCOMP, 0, -1 );
  % repetir enquanto nao FDA ( ARQCOMP )
     for ( Y = 1 : Linhas )
     % guardar a posicao antes de ler
        P1 = ftell ( ARQCOMP );
     % 3.1 ler um complexo
        X.Re = fscanf ( ARQCOMP, '%f', 1 );
        X.Im = fscanf ( ARQCOMP, '%f', 1 );
     % guardar a posicao apos ler
        P2 = ftell ( ARQCOMP );
     % 3.2 mostrar um complexo
        printf ( '\n(%f, %f)', X.Re, X.Im );
     % 3.3 alterar complexo
        Z = input ( '\nAlterar (Sim=1, Nao=0) ? ' );
        if ( Z == 1 )
        % 3.3.1 ler novo complexo
           X.Re = input ( '\nQual a parte real ? ' );
           X.Im  = input ( '\nQual a parte imaginaria ? ' );
        % 3.3.2 regravar complexo
           % reposicionar na linha adequada
              STATUS = fseek ( ARQCOMP, P1, -1 );
           % gravar na mesma posicao
               printf ( '%f %f\n', X.Re, X.Im );
        end % if
     % reposicionar para ler
        STATUS = fseek ( ARQCOMP, P2+1, -1 );
     end % for
 % 4. fechar arquivo
    fclose ( ARQCOMP );

% pausa para terminar
   printf  ( ' \n\nPressionar qualquer tecla para terminar.' );
   pause;
% fim do programa
```

Programa em C++:

```
// Exemplo 5a.
// Ler arquivo texto e alterar complexos.
//
// bibliotecas necessarias
#include <fstream>            // para arquivos
#include <iostream>
using namespace std;
//
// definir tipos globais
//
  typedef struct
  {                            // registro
    float Re;                  //   parte real
    float Im;                  //   parte imaginaria
  } COMPLEXO;                  // fim registro
//
// definir metodos
//
int TAMANHO ( char Nome [ ] )
{
 // P.1 - definir dado local
  int              Linhas = 0;
  COMPLEXO X;
  fstream        A;
// P.2 - abrir arquivo
  A.open ( Nome, ios::in );
// P.3 - ler e contar linhas
  // ler primeira linha
     A >> X.Re >> X.Im;
  // repetir enquanto nao FDA ( A )
     while ( ! A.eof ( ) )
     {
      // contar mais uma linha
         Linhas = Linhas + 1;
      // tentar ler outra linha
        A >> X.Re >> X.Im;
     } // fim repetir para
  return ( Linhas );
} // fim da funcao TAMANHO ( )
```

```
//
// parte principal
//
int main (void)
{
 // 1. definir dados
    COMPLEXO X;
    fstream        ARQCOMP;
    int            Linhas, Y, Z;
    long           P1, P2;
 // 2. abrir arquivo para alteracao
    ARQCOMP.open ( "COMPLEXOS.TXT", ios::in | ios::out );
 // 3. ler dados de arquivo
    // repetir enquanto nao fim de arquivo
       Linhas = TAMANHO ( "COMPLEXOS.TXT" );
       for ( Y = 1; Y <= Linhas; Y = Y + 1 )
       { // guardar a posicao antes de ler
            P1 = ARQCOMP.tellp ( );
         // 3.1 ler partes de um complexo
            ARQCOMP >> X.Re >> X.Im;
         // guardar a posicao apos ler
            P2 = ARQCOMP.tellp ( );
         // 3.2 mostrar um complexo
            cout << "(" << X.Re << "," << X.Im << ")";
         // 3.3 alterar complexo
            cout << "\nAlterar (Sim=1, Nao=0) ? ";
            cin  >> Z;
            if ( Z == 1 )
            { // 3.3.1 ler um novo complexo
                cout << "\nQual a parte real ? ";
                cin   >> X.Re;
                cout << "\nQual a parte imaginaria ? ";
                cin   >> X.Im;
              // 3.3.2 regravar complexo
                // reposicionar na linha adequada
                   ARQCOMP.seekg ( P1, ios::beg );
                // gravar na mesma posicao
                   ARQCOMP << "\n" << X.Re << " " << X.Im;
            } // fim if
         // reposicionar para ler
            ARQCOMP.seekg ( P2, ios::beg );
       } // fim while // 4. fechar arquivo
   ARQCOMP.close ( );
// pausa para terminar
   cout << "\nPressionar qualquer tecla para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Outra versão em C++:

```
// Exemplo 5b.
// Ler arquivo binario e alterar complexos.
//
// bibliotecas necessarias
#include <fstream>            // para arquivos
#include <iostream>
using namespace std;
//
// definir tipos globais
//
  typedef struct
  {                            // registro
    float Re;                  //   parte real
    float Im;                  //   parte imaginaria
  } COMPLEXO;                  // fim registro
//
// definir metodos
//
int TAMANHO ( char Nome [ ] )
{
 // P.1 - definir dado local
  int              Linhas = 0;
  COMPLEXO X;
  fstream        A;
// P.2 - abrir arquivo
  A.open ( Nome, ios::binary | ios::in );
// P.3 - ler e contar linhas
  // ler primeiro complexo
     A.read ( (char *) &X, sizeof ( COMPLEXO ) );
  // repetir enquanto nao FDA ( A )
     while ( ! A.eof ( ) )
     {
      // contar mais uma linha
         Linhas = Linhas + 1;
      // tentar ler outro complexo
         A.read ( (char *) &X, sizeof ( COMPLEXO ) );
     } // fim repetir para
  return ( Linhas );
} // fim da funcao TAMANHO ( )
```

```
// parte principal
//
int main (void)
{
 // 1. definir dados
    COMPLEXO X;
    fstream         ARQCOMP;
    int               Linhas, Y, Z;
 // 2. abrir arquivo para alteracao
    ARQCOMP.open ( "COMPLEXOS.TXT", ios::binary | ios::in | ios::out );
 // 3. ler dados de arquivo
    // repetir enquanto nao fim de arquivo
       Linhas = TAMANHO ( "COMPLEXOS.TXT" );
       for ( Y = 1; Y <= Linhas; Y = Y + 1 )
       { // posicionar para ler
            ARQCOMP.seekg ( ( Y-1 )* sizeof ( COMPLEXO ), ios::beg );
         // 3.1 ler partes de um complexo
            ARQCOMP.read ( (char *) &X, sizeof ( COMPLEXO ) );
         // 3.2 mostrar um complexo
            cout << "(" << X.Re << "," << X.Im << ")";
         // 3.3 alterar complexo
            cout << "\nAlterar (Sim=1, Nao=0) ? ";
            cin  >> Z;
            if ( Z == 1 )
            { // 3.3.1 ler um novo complexo
                 cout << "\nQual a parte real ? ";
                 cin  >> X.Re;
                 cout << "\nQual a parte imaginaria ? ";
                 cin  >> X.Im;
              // 3.3.2 regravar complexo
                 // reposicionar
                    ARQCOMP.seekg ( ( Y-1 )* sizeof ( COMPLEXO ), ios::beg );
                 // gravar na mesma posicao
                    ARQCOMP.write   ( (char *) &X, sizeof ( COMPLEXO ) );
            } // fim if
       } // fim while // 4. fechar arquivo
   ARQCOMP.close ( );
// pausa para terminar
   cout << "\nPressionar qualquer tecla para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Programa em C#:

```
/*
 * Exemplo 5
 * Ler arquivo texto e alterar complexos.
 */
using System;
using System.IO;

class COMPLEXO
{
  public double Re = 0.0;            // parte real
  public double Im = 0.0;            // parte imaginaria
} // fim COMPLEXO class

class Exemplo_5
{
   public static int TAMANHO ( string Nome )
   {
   // definir dado local
      int       Linhas = 0;
      string   Linha;
   // abrir arquivo
      TextReader A = new StreamReader ( Nome );
   // ler e contar linhas
      // ler primeira linha
         Linha  = A.ReadLine ( );
      // repetir enquanto nao FDA ( A )
         while ( Linha != null )
         {
         // contar mais uma linha
            Linhas = Linhas + 1;
         // tentar ler outra linha
            Linha = A.ReadLine ( );
         } // fim while
   // 4. fechar arquivo
      A.Close ( );
      return ( Linhas );
   } // fim da funcao TAMANHO ( )
```

```
//
// parte principal
//
   public static void Main ( )
   {
   // 1. definir dados
      COMPLEXO X = new COMPLEXO ( );
      int      Linhas,
               Y, Z;
      string Linha;
   // 2. abrir arquivo para leitura
      TextReader ARQCOMP = new StreamReader ( "COMPLEXOS.TXT" );
      TextWriter ARQTEMP = new StreamWriter ( "COMPLEXOS.$$$" );
   // 3. ler dados de arquivo
         Linhas = TAMANHO ( "COMPLEXOS.TXT" );
      // repetir enquanto nao fim de arquivo
         for ( Y = 1; Y <= Linhas/2; Y = Y + 1 )
         {
         // 3.1 ler partes de um complexo
            Linha = ARQCOMP.ReadLine ( );
            X.Re  = double.Parse ( Linha );
            Linha = ARQCOMP.ReadLine ( );
            X.Im  = double.Parse ( Linha );
         // 3.2 mostrar um complexo
            Console.Write ( "\n(" + X.Re + "," + X.Im + ")" );
         // 3.3 alterar complexo
            Console.Write ( "\nAlterar (Sim=1, Nao=0) ? " );
            Z = int.Parse ( Console.ReadLine ( ) );
            if ( Z == 1 )
            { // 3.3.1 ler um novo complexo
                Console.Write ( "\nQual a parte real ? " );
                X.Re = int.Parse ( Console.ReadLine ( ) );
                Console.Write ("\nQual a parte imaginaria ? " );
                X.Im = int.Parse ( Console.ReadLine ( ) );
            } // fim if
            // 3.3.2 regravar complexo
               ARQTEMP.WriteLine ( X.Re );
               ARQTEMP.WriteLine ( X.Im  );
         } // fim for
         ARQCOMP.Close ( );
         ARQTEMP.Close  ( );
      // trocar nome e remover o temporario
         if ( File.Exists ( "COMPLEXOS.$$$" ) )
         {
           File.Delete ( "COMPLEXOS.TXT" );
           File.Copy   ( "COMPLEXOS.$$$", "COMPLEXOS.TXT" );
           File.Delete ( "COMPLEXOS.$$$" );
         } // fim se

   // pausa para terminar
      Console.Write       ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_5 class
```

Programa em Java:

```
/**
  * Exemplo 5
  * Ler arquivo texto e alterar complexos.
  */

// ------------------------------------------------ classes necessarias
import IO.*;                              // IO.jar   deve ser acessivel
// ------------------------------------------------ definicao de classe

class COMPLEXO
{
  public double Re = 0.0;            // parte real
  public double Im = 0.0;            // parte imaginaria
} // fim COMPLEXO class

class Exemplo_5
{
   public static int TAMANHO ( String Nome )
   {
   // definir dado local
      COMPLEXO X = new COMPLEXO ( );
      int               Linhas = 0;
      String           Linha;
   // abrir arquivo
      FILE A = new FILE ( FILE.INPUT, Nome );
   // ler e contar linhas
      // ler primeira linha
         Linha = A.readln ( );
      // repetir enquanto não FDA ( A )
         while ( ! A.eof ( ) )
         {
         // contar mais uma linha
            Linhas = Linhas + 1;
         // tentar ler outra linha
            Linha = A.readln ( );
         } // fim while
   // 4. fechar arquivo
      A.close ( );
      return ( Linhas );
   } // fim da funcao TAMANHO ( )
```

```
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 1. definir dados
      COMPLEXO X = new COMPLEXO ( );
      int        Linhas,
                 Y, Z;
      String  Linha;
   // 2. abrir arquivo para leitura
      FILE ARQCOMP = new FILE ( FILE.INPUT ,   "COMPLEXOS.TXT" );
      FILE ARQTEMP  = new FILE ( FILE.OUTPUT, "COMPLEXOS.$$$" );
   // 3. ler dados de arquivo
         Linhas = TAMANHO ( "COMPLEXOS.TXT" );
      // repetir enquanto nao fim de arquivo
         for ( Y = 1; Y <= Linhas; Y = Y + 1 )
         {
         // 3.1 ler partes de um complexo
            Linha = ARQCOMP.read ( );
            X.Re  = Double.parseDouble ( Linha );
            Linha = ARQCOMP.read ( );
            X.Im  = Double.parseDouble ( Linha );
         // 3.2 mostrar um complexo
            IO.println ( "\n(" + X.Re + "," + X.Im + ")" );
         // 3.3 alterar complexo
            Z = IO.readint ( "\nAlterar (Sim=1, Nao=0) ? " );
            if ( Z == 1 )
            { // 3.3.1 ler um novo complexo
                X.Re = IO.readdouble ( "\nQual a parte real ? " );
                X.Im  = IO.readdouble ( "\nQual a parte imaginaria ? " );
            } // fim if
            // 3.3.2 regravar complexo
               ARQTEMP.println ( "" + X.Re + " " + X.Im );
         } // fim for
         ARQCOMP.close   ( );
         ARQTEMP.close    ( );
         ARQCOMP.delete  ( );
         ARQTEMP.rename ( "COMPLEXOS.TXT" );
   // pausa para terminar
      IO.pause ( "\nPressionar ENTER para terminar." );
   } // end main ( )

} // fim Exemplo_5 class
```

Exercícios propostos.

1. Fazer um algoritmo para:
   - definir um registro contendo hora, minutos e segundos;
   - ler duas indicações de tempo diferentes e calcular e imprimir a diferença entre elas.

2. Fazer um algoritmo para:
   - definir um registro para representar um número complexo na forma cartesiana;
   - ler dois números complexos diferentes;
   - calcular e mostrar: a soma, a diferença, o produto, o quociente, a norma e o ângulo

3. Fazer um algoritmo para:
   - definir um registro para representar um número complexo na forma polar;
   - ler dois números complexos diferentes; calcular e mostrar: a soma, a diferença e o produto

4. Fazer um algoritmo para:
   - definir um registro para conter dados de um aluno: número, nome, número de disciplinas cursadas no semestre, lista com até 10 disciplinas;
   - gravar um arquivo com os dados de uma turma, cujo último aluno terá o número 99999.

5. Fazer um algoritmo para:
   - ler o arquivo do problema anterior;
   - calcular e mostrar: quantos alunos matricularam em mais de 03 disciplinas; o maior número de disciplinas cursadas; os alunos com o menos de 03 disciplinas cursadas.

6. Fazer um algoritmo para:
   - inserir mais 05 alunos na turma pela ordem do seu número.

7. Fazer um algoritmo para:
   - copiar o arquivo anterior, removendo 02 alunos cujos números serão lidos pelo teclado.

8. Fazer um algoritmo para:
   - ler o arquivo anterior e gravar um outro arquivo, indexando-o por ordem alfabética.

9. Fazer um algoritmo para:
   - ler o arquivo do problema anterior e acrescentar mais 03 alunos na ordem alfabética.

10. Fazer um algoritmo para:
    - remover do arquivo do problema anterior 07 alunos cujos números serão lidos do teclado.

11. Fazer um algoritmo para:
    - ler um arquivo contendo valores inteiros; o primeiro valor é a quantidade de inteiros;
    - separar em dois arquivos diferentes os pares e os ímpares.

12. Fazer um algoritmo para:
    - ler um arquivo contendo valores inteiros; o primeiro valor é a quantidade de inteiros;
    - separar em dois arquivos diferentes os pares e positivos, e os ímpares e negativos.

13. Fazer um algoritmo para:
    - gravar em um arquivo os 64 primeiros valores da série: 1 1 2 3 5 8 13 21 39 . . .

14. Fazer um algoritmo para:
    - ler o arquivo do problema anterior e armazenar em um vetor apenas os primos.

15. Fazer um algoritmo para:
    - ler o arquivo do problema (13) e armazenar em um vetor apenas os primos.

16. Fazer um algoritmo para:
    - ler o arquivo do problema (13) e armazenar os dados em uma matriz 8 x 8.