

Notas de Aula de Algoritmos e Programação de Computadores

FLÁVIO KEIDI MIYAZAWA

com a colaboração de

TOMASZ KOWALTOWSKI

Instituto de Computação - UNICAMP

Versão 2000.1

Estas notas de aula não devem ser usadas como única fonte de estudo. O aluno deve ler outros livros disponíveis na literatura.

Nenhuma parte destas notas pode ser reproduzida, qualquer que seja a forma ou o meio, sem a permissão dos autores.

Os autores concedem a permissão explícita para a utilização e reprodução deste material no contexto do ensino de disciplinas regulares dos cursos de graduação sob a responsabilidade do Instituto de Computação da UNICAMP.

© Copyright 2000

Instituto de Computação
UNICAMP
Caixa Postal 6176
13083-970 Campinas-SP
{fkm,tomasz}@ic.unicamp.br

12 Passagem de funções e procedimentos como parâmetros

A linguagem pascal permite passar rotinas como parâmetros. Isto nos possibilita fazer rotinas focalizadas em um método de forma bem mais genérica usando rotinas particulares para cada tipo de dado, que são passadas como parâmetros.

Mas antes de apresentar a sintaxe destas declarações de parâmetros, precisamos fazer algumas observações entre dois dos principais padrões da linguagem Pascal: o do Pascal Extendido e o Borland Pascal. Existem algumas diferenças entre estes dois padrões e a passagem de funções e procedimentos como parâmetros é uma delas.

12.1 Diferenças entre Borland/Turbo Pascal e Extended Pascal

Borland Pascal O padrão Borland Pascal apresenta declaração de novos tipos associados a procedimentos ou funções, mas não aceita a descrição do tipo do procedimento ou função na declaração do parâmetro. Assim, para declarar um parâmetro que é um procedimento ou função precisamos primeiro definir tipos associados a estas rotinas e só depois declarar as rotinas como parâmetros usando estes tipos.

type

IdentificadorTipoFuncao = **function**(Lista_de_Parâmetros): Tipo_Returno_Função;

IdentificadorTipoProcedure = **procedure**(Lista_de_Parâmetros);

O tipo da rotina é igual a declaração do cabeçalho da rotina, tirando o identificador da rotina. Com isso, podemos declarar um parâmetro que é uma rotina como um parâmetro normal.

Obs.: No Turbo Pascal é necessário colocar a opção de funções e procedimentos com endereçamento **far** (Options, Compiler, Force Far Calls: On) ou colocar a diretiva {**\$F+**} antes da declaração das rotinas. No Free Pascal é necessário compilar o programa com a opção de **-So** na linha de comando.

Extended Pascal O padrão Extended Pascal não apresenta declaração de novos tipos como sendo procedimentos ou funções. Estes parâmetros devem ser declarados diretamente na declaração do parâmetro.

As sintaxes das declarações de parâmetro como procedimento ou função é a mesma que a usada nos cabeçalhos de funções.

Exemplo: `procedure P(function F(x: real): real);`

No cabeçalho do procedimento P, temos a declaração de um parâmetro chamado F que é uma função. F é uma função que tem um parâmetro real e retorna um valor real.

O padrão do pascal extendido (*Extended Pascal standard*) foi completado em 1989 e é tanto um padrão ANSI/IEEE como ISO/IEC.

Exemplo 12.1 *No exemplo das figuras 38 e 39, temos um procedimento chamado imprime, com 4 parâmetros: dois números reais (a e b), uma função (fun) e uma string (msg). O procedimento imprime um texto que contém os dois valores reais e o resultado da chamada da função sobre estes dois números. Note que esta rotina poderia ter sido chamada com qualquer função que tem dois parâmetros com valores reais e retorna um valor real. No caso, a rotina imprime foi chamada com as funções maximo, minimo e media.*

Note que na declaração, foram colocados identificadores associados aos parâmetros. A declaração destes tipos necessita que tenhamos identificadores, mas seu nome não precisa estar associado aos nomes dos identificadores na descrição da rotina.

A tradução de programas escritos em Extended Pascal para Borland Pascal é simples, uma vez que precisamos apenas mudar a declaração de rotinas que são parâmetros, de forma a usar tipos. Já a tradução de um programa escrito em Borland Pascal para Extended Pascal pode ser mais complicada, se aquele usar os tipos de rotinas dentro de estruturas como vetores e registros.

A partir de agora usaremos apenas uma destas sintaxes. Escolhemos a sintaxe do Borland Pascal, uma vez que esta sintaxe pode ser usada na maioria dos compiladores Pascal, tais como: Turbo Pascal (versão 5 em diante), Borland Pascal, Delphi, gpc (Gnu Pascal Compiler) e Free Pascal. Acreditamos que futuras padronizações da linguagem Pascal venham a contemplar este tipo de sintaxe.

```

program ParametroFuncaoBorlandPascal;
type TipoFuncao = function(a,a : real):real;
    TipoMsg = string[100];
function maximo(a,b : real):real;
begin
    if (a>b) then maximo := a
    else maximo := b;
end;
function minimo(a,b : real):real;
begin
    if (a>b) then minimo := b
    else minimo := a;
end;
function media(a,b : real):real;
begin
    media := (a+b)/2;
end;
procedure imprime(a,b : real; fun: tipoFuncao;
    msg : TipoMsg);
begin
    writeln(msg,' de ',a:7:2,' e ',b:7:2,' é: ',
    fun(a,b):7:2);
end;

var n1,n2 : real;
begin
    write('Entre com dois números: ');
    readln(n1,n2);
    imprime(n1,n2,maximo,'O máximo');
    imprime(n1,n2,minimo,'O mínimo');
    imprime(n1,n2,media,'A média');
end.

```

Figura 38: Exemplo de rotinas como parâmetros em Borland Pascal

```

program ParametroFuncaoExtendedPascal;
type TipoMsg = string[100];
function maximo(a,b : real):real;
begin
    if (a>b) then maximo := a
    else maximo := b;
end;
function minimo(a,b : real):real;
begin
    if (a>b) then minimo := b
    else minimo := a;
end;
function media(a,b : real):real;
begin
    media := (a+b)/2;
end;
procedure imprime(a,b : real;
    function fun(a,a : real):real;
    msg : TipoMsg);
begin
    writeln(msg,' de ',a:7:2,' e ',b:7:2,' é: ',
    fun(a,b):7:2);
end;

var n1,n2 : real;
begin
    write('Entre com dois números: ');
    readln(n1,n2);
    imprime(n1,n2,maximo,'O máximo');
    imprime(n1,n2,minimo,'O mínimo');
    imprime(n1,n2,media,'A média');
end.

```

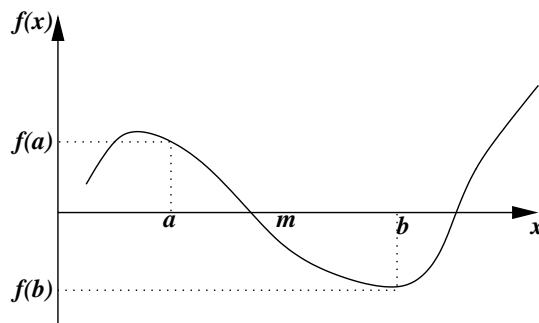
Figura 39: Exemplo de rotinas como parâmetros em Extended Pascal

12.2 Método de bisseção para encontrar raízes de funções

Um método para encontrar soluções de uma função é através do método de bisseção. Para usar estes método devemos ter:

- Uma função $f(x)$ para poder calcular um valor de x_0 tal que $f(x_0) = 0$.
- Um intervalo de busca da solução $[a, b]$, de tal forma que $f(a) \cdot f(b) \leq 0$.

Note que o segundo item nos garante que ou a é solução, ou b é solução ou deve existir uma solução x_0 no intervalo $[a, b]$ tal que $f(x_0) = 0$ (veja a figura seguinte).



A idéia é dividir este intervalo em duas partes pelo meio, m , digamos $[a, m]$ e $[m, b]$. Devemos ter pelo menos uma das condições seguintes:

1. $f(a) \cdot f(m) \leq 0$
2. $f(m) \cdot f(b) \leq 0$.

Se $f(a) \cdot f(m) \leq 0$ então necessariamente temos uma solução dentro do intervalo $[a, m]$ e repetimos o mesmo processo, desta vez interagando com o intervalo $[a, m]$ no lugar do intervalo $[a, b]$. Caso contrário, teremos uma solução dentro do intervalo $[m, b]$ e continuamos o processo neste intervalo. A cada interação o tamanho do intervalo diminui para metade do tamanho do intervalo anterior. Assim, o processo pode parar quando o comprimento do intervalo for suficientemente pequeno.

Com isto, podemos escrever uma rotina que encontra uma solução para uma certa função (recebida pela rotina como parâmetro) e o intervalo de atuação.

O método de bisseção já pressupõe um intervalo $[a, b]$ válido tal que $f(a) \cdot f(b) \leq 0$. Para encontrar tal intervalo, podemos começar com um intervalo suficientemente grande $[a_0, a_k]$, dividimos este intervalo em partes pequenas, digamos em k partes iguais, $[a_0, a_1], [a_1, a_2], \dots, [a_{k-1}, a_k]$ e verificamos se algum destes intervalos digamos $[a_{i-1}, a_i]$, $i \in \{1, \dots, k\}$ é tal que $f(a_i) \cdot f(a_{i+1}) \leq 0$. Caso exista tal intervalo, usamos o método da bisseção no intervalo, caso contrário terminamos sem sucesso. Naturalmente podem ocorrer instâncias onde este método não encontra soluções, mesmo que tal solução se encontre em um dos intervalos iniciais. Além disso, dificilmente este método irá encontrar soluções de funções não negativas ou não positivas.

O programa a seguir apresenta a implementação do método descrito acima.

```

program ProgramaBicessao;
const EPS = 0.000000001;
type TipoFuncaoReal = function (x:real):real;
{Encontrando o intervalo, retorna
sucesso=true e a0 e ak atualizados com
o intervalo, caso contrário, retorna com
sucesso=false}
procedure EncontraIntervalo(f: TipoFuncaoReal ;
    var a0,ak: real;k:integer;
    var sucesso: boolean);
var delta,a: real;
begin
    a:=a0;
    delta := (ak-a)/k;
    while (a < ak) and (f(a)*f(a+delta)>0)
        do a := a + delta;
    if (f(a)*f(a+delta)>0) then sucesso := false
    else begin
        a0 := a;
        ak := a+delta;
        sucesso := true;
    end;
end;

```

```

{Encontrando solução, retorna sucesso=true e
a raiz em solução, cc. sucesso=false}
procedure bicessao(f: TipoFuncaoReal; a,b:real;
    var sucesso:boolean;var solucao:real);
var m : real;
begin
    EncontraIntervalo(f,a,b,1000,sucesso);
    if (sucesso) then begin
        while (abs(a-b)>EPS) do begin
            m := (a+b)/2;
            if (f(a)*f(m)<=0) then b:=m
            else a:=m;
        end;
        solucao := m;
    end;
end;

```

```

{função real:  $f(x)=x^3 + 2x^2 - x + 10$  }
function f1(x : real):real;
begin
    f1 := x*x*x + 2*x*x - x + 10;
end; { f1 }

```

```

{função real:  $f(x)=(x-5)*(x-2)*(x-3)*(x-4)*(x-6)$ }
function f2(x : real):real;
begin
    f2 := (x-5)*(x-2)*(x-3)*(x-4)*(x-6);
end; { f1 }

var sucesso : boolean;
    raiz: real;
begin
    bicessao(f1,-10,10,sucesso,raiz);
    if (sucesso) then
        writeln('Uma solução de  $f(x)=x^3+2x^2-x+10$ 
        é: ',
            raiz:10:8);
    bicessao(f2,-10,10,sucesso,raiz);
    if (sucesso) then
        writeln('Uma solução de ',
            '  $f(x)=(x-5)*(x-2)*(x-3)*(x-4)*(x-6)$ 
            é: ',
            raiz:10:8);
end.

```

Exercício 12.1 (*Integral de funções*) O cálculo da integral de uma curva definida f pode ser calculado por aproximações utilizando-se o método dos trapézios. Neste método dividimos o intervalo de integração em n partes iguais, digamos divididas pelos pontos x_0, x_1, \dots, x_n e calculamos a área do trapézio substituindo a curva em cada intervalo x_{i-1}, x_i por uma reta ligando os pontos $f(x_{i-1})$ e $f(x_i)$, $i = 1, \dots, n$ (veja a figura seguinte). O valor aproximado da integral definida é a soma das áreas dos n trapézios.

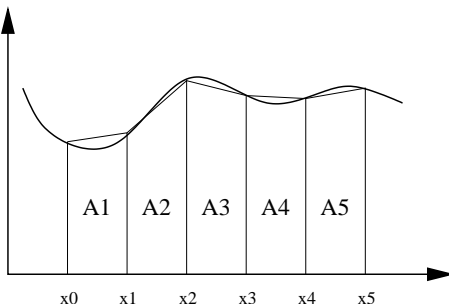
$$\int_a^b f(x) dx \sim \sum_{i=1}^n A_i, \quad \text{onde}$$

$$A_i = \frac{f(x_{i-1}) + f(x_i)}{2} \cdot \delta x, \quad i = 1, \dots, n \quad \text{Área de cada trapézio}$$

$$\delta x = x_i - x_{i-1} = \frac{b - a}{n}$$

$$x_0 = a$$

$$x_n = b.$$



Naturalmente, quanto maior o valor de n , melhor a aproximação da integral. Faça um programa contendo uma função chamada *integral* com o seguinte cabeçalho:

function *integral*(f : *TipoFuncaoReal*; a, b : **real**; n : **integer**): **real**;

onde *TipoFuncaoReal* é um tipo de função como declarado como programa de bicesão; os valores a e b definem o intervalo de integração: $[a, b]$; e o valor n indica a quantidade de intervalos a ser usado no método de aproximações.

12.3 Ordenação usando funções de comparação

No exemplo 6.9 vimos como podemos ordenar um vetor colocando na posição correta o maior elemento não considerado a cada instante. Note que o algoritmo faz a seleção do maior elemento fazendo várias comparações entre dois elementos (se menor, igual ou maior). Em muitos problemas, cada elemento do vetor contém informações de um objeto que contém em geral várias características e estes podem ser ordenados de várias maneiras, dependendo das características consideradas.

Por exemplo, vamos considerar que uma universidade, chamada UNICOMP, armazena os dados dos alunos usando o tipo *TipoAluno*, apresentado na página 90. Por simplicidade, vamos considerar os campos *Nome*, *DataNascimento* e *CR*. Para efeitos de busca e classificação, a universidade necessita que estes dados possam estar em ordem alfabética pelo nome ou listados pelos alunos mais novos primeiro ou listados ordenados pelo CR em ordem decrescente. Como é apenas a comparação entre elementos que muda, podemos fazer apenas uma rotina de ordenação e usar diferentes funções de comparação para decidir qual aluno vem antes de outro.

A função para comparar dois elementos tem o seguinte formato:

function *Compara*(**var** a, b : *TipoAluno*):**char**;

a função retorna '<', '=' ou '>', se a é menor, igual ou maior que b , respectivamente. A seguinte função compara os nomes de dois alunos:

```
function ComparaNome(Aluno1, Aluno2 : TipoAluno):char;
var Nome1, Nome2      : TipoNome;
begin
  if (Aluno1.Nome < Aluno2.Nome) then ComparaNome := '<'
  else if (Aluno1.Nome > Aluno2.Nome) then ComparaNome := '>'
  else ComparaNome := '=';
end;
```

A seguir, vamos descrever a nova rotina de ordenação (atualizada a partir do exemplo 7.13) que pode usar a função de comparação por nome, dado acima.

```

const MAX = 100;
type { --> Incluir tipos em TipoAluno <-- }
  TipoVetorAluno = array[1..MAX] of TipoAluno;
  TipoFuncaoComparacao = function (a,b: TipoAluno):char;
procedure SelectionSortAluno(fcomp: TipoFuncaoComparacao; var v : TipoVetorAluno; n:integer);
var m,imax : integer;
procedure TrocaAluno(var Aluno1, Aluno2 : TipoAluno);
  var AlunoAux : TipoAluno;
  begin AlunoAux:=Aluno1; Aluno1:=Aluno2; Aluno2:=AlunoAux; end;

function IndMaxAluno(fcomp : TipoFuncaoComparacao; var v:TipoVetorAluno; n:integer):integer;
var i, Ind : integer;
begin
  if (n <=0) then Ind := 0
  else begin
    Ind := 1; {O maior elemento começa com o primeiro elemento do vetor}
    for i:=2 to n do if (fcomp(v[Ind],v[i])= ' < ') then Ind:=i;
  end;
  IndMaxAluno := Ind;
end;
begin
  for m:=n downto 2 do begin
    imax := IndMaxAluno(fcomp,v,m);
    TrocaAluno(v[m],v[imax]);
  end;
end;

```

Note que a mudança (além do tipo de dado) do programa apresentado no exemplo 6.9 está no uso da função de comparação entre dois elementos, que é um parâmetro usado na chamada da rotina *SelectionSortAluno*. Assim, a chamada da rotina *SelectionSortAluno(ComparaAluno, V, n)* fará a ordenação do vetor *V*, com *n* elementos, usando a função de comparação *ComparaAluno*. I.e., o vetor ficará em ordem alfabética pelo nome.

Agora, para fazer a ordenação em ordem crescente de idade (mais novos primeiro), podemos usar o mesmo programa usando apenas uma função de comparação diferente. Mas note que agora queremos os de maior data de nascimento primeiro. Para isso, vamos “*tapear*” a rotina de ordenação, usando uma função de comparação que responde dizendo que o aluno A_1 é *menor* (‘<’) que o aluno A_2 se a data de nascimento de A_1 for na verdade *maior* que a de A_2 , e vice versa. Isto fará com que a rotina ordene de maneira invertida. Lembre que a rotina de ordenação foi feita de maneira que os menores ficassem nas primeiras posições; assim, se usamos a rotina de comparação que diz que um elemento é menor quando na verdade é maior, e vice-versa, obteremos uma ordenação com os maiores nas primeiras posições. A rotina de comparação de alunos por data de nascimento (mais novos primeiro) pode ser dada pela seguinte função:

```

function ComparaIdade(Aluno1,Aluno2 : TipoAluno):char;
begin
  if (Aluno1.DataNascimento.Ano < Aluno2.DataNascimento.Ano) then ComparaIdade := ' > '
  else if (Aluno1.DataNascimento.Ano > Aluno1.DataNascimento.Ano) then ComparaIdade := ' < '
  else if (Aluno1.DataNascimento.Mes < Aluno1.DataNascimento.Mes) then ComparaIdade := ' > '
  else if (Aluno1.DataNascimento.Mes > Aluno1.DataNascimento.Mes) then ComparaIdade := ' < '
  else if (Aluno1.DataNascimento.Dia < Aluno1.DataNascimento.Dia) then ComparaIdade := ' > '
  else if (Aluno1.DataNascimento.Dia > Aluno1.DataNascimento.Dia) then ComparaIdade := ' < '
  else ComparaIdade := ' = ' ;
end;

```

Note também que o uso da função de comparação permite fazer comparações de elementos levando em consideração vários dados para desempate. O programa final é dado a seguir.

```

program OrdenaAlunoIdadeNome;
const MAX = 100;
type { --> Incluir tipos em TipoAluno <-- }
  TipoVetorAluno = array[1..MAX] of TipoAluno;
  TipoFuncaoComparacao = function (a,b: TipoAluno):char;
function ComparaNome(Aluno1,Aluno2 : TipoAluno):char;
var Nome1,Nome2      : TipoNome;
begin
  if (Aluno1.Nome < Aluno2.Nome) then ComparaNome := '<'
  else if (Aluno1.Nome > Aluno2.Nome) then ComparaNome := '>'
  else ComparaNome := '=';
end; { ComparaNome }
function ComparaIdade(Aluno1,Aluno2 : TipoAluno):char;
begin { Inserir código da função ComparaIdade, visto anteriormente. } end;

procedure SelectionSortAluno(fcomp: TipoFuncaoComparacao; var v : TipoVetorAluno; n:integer);
var m,imax : integer;
  procedure TrocaAluno(var Aluno1, Aluno2 : TipoAluno);
    var AlunoAux : TipoAluno;
    begin AlunoAux:=Aluno1; Aluno1:=Aluno2; Aluno2:=AlunoAux; end;
  function IndMaxAluno(fcomp : TipoFuncaoComparacao; var v:TipoVetorAluno; n:integer):integer;
    var i, Ind : integer;
    begin
      if (n <=0) then Ind := 0
      else begin
        Ind := 1; { O maior elemento começa com o primeiro elemento do vetor }
        for i:=2 to n do if (fcomp(v[Ind],v[i])='<') then Ind:=i;
      end;
      IndMaxAluno := Ind;
    end;
begin
  for m:=n downto 2 do begin
    imax := IndMaxAluno(fcomp,v,m);
    TrocaAluno(v[m],v[imax]);
  end;
end;
procedure ImprimeVetorAluno(var v : TipoVetorAluno; n:integer; msg:TipoString);
begin { Exercício: Rotina para imprimir os Alunos na tela } end;
procedure LeVetorAluno(var v:TipoVetorAluno;var n:integer);
begin { Exercício: Rotina para ler Alunos } end;

var i, n, Ind : integer;
  V      : TipoVetorFunc;
begin
  LeVetorAluno(V,n);
  SelectionSortFunc(comparaNome,v,n);
  ImprimeVetorFunc(v,n,'Ordenados por nome (ordem alfabética)');
  SelectionSortFunc(comparaIdade,v,n);
  ImprimeVetorFunc(v,n,'Ordenados por idade (mais novos primeiro)');
end.

```

Exercício 12.2 A universidade UNICOMP deseja dar bolsas de estudo para os alunos que apresentam melhores valores do Coeficiente de Rendimento (CR). Assim, faça uma função de comparação para ordenar os alunos, colocando os de maiores valores do CR primeiro. Caso dois alunos possuam o mesmo valor de CR, estes devem ser desempatados pelo RA.

12.4 Exercícios

- É dado uma função $f : \mathbb{R} \rightarrow \mathbb{R}$ e um intervalo $[a, b]$ tal que $f(a) \cdot f(b) < 0$. Faça uma função recursiva que encontra uma solução $x \in [a, b]$ usando o método da bissecção (dividindo sucessivamente o intervalo ao meio) tal que $f(x) = 0$. A função pode dar o resultado dentro de uma precisão de 0,0001 (diferença entre a solução verdadeira e a solução encontrada pela rotina) e deve ter o seguinte cabeçalho:

function Bissecao(f : TipoFuncao; a, b : real): real;

onde f é a função para a qual calcularemos a solução e $[a, b]$ define o intervalo que contém a solução. O tipo *TipoFuncao* é definido como:

type TipoFuncao = function (x:real):real;

- Para calcular a integral $S = \int_a^b f(x) dx$ pode-se usar a aproximação de uma soma finita de “valores amostrados”:

$$S_k = \frac{h}{3}(f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \dots + 4f_{n-3} + 2f_{n-2} + 4f_{n-1} + f_n)$$

onde

$$f_i = f(a + ih), \quad h = (b - a)/n, \quad e \quad n = 2^k.$$

O número de pontos de amostragem é $n + 1$ e h é a distância entre dois pontos de amostragem adjacentes. O valor S da integral é aproximado pela seqüência S_1, S_2, S_3, \dots , que converge se a função é suficientemente bem comportada (suave). O método acima é chamado de *método de Simpson*.

Escreva uma função que calcule a integral de uma função $f(x)$ usando o método de Simpson. A função tem como parâmetros: a função $f(x)$, o intervalo de integração $[a, b]$ e o valor k . A função retorna a aproximação da integral de $f(x)$ no intervalo $[a, b]$. Aplique sua função para calcular as integrais:

$$\int_0^2 x^2 dx \quad e \quad \int_0^{\pi/2} \text{sen}(x) dx$$

- Faça um programa para manipular um cadastro de alunos, nos moldes do exemplo da seção 12.3, de maneira a ter as seguintes opções:
 - Iniciar cadastro vazio (inicialmente o cadastro já começa vazio (número de elementos igual a zero)).
 - Inserir um novo elemento no cadastro (se o cadastro estiver cheio, avise que não há memória disponível).
 - Ordenar o cadastro por nome em ordem alfabética.
 - Ordenar o cadastro por idade, mais novos primeiro.
 - Ordenar o cadastro por idade, mais velhos primeiro.
 - Ordenar o cadastro por RA, ordem crescente.
 - Ordenar o cadastro por CR, maiores primeiro.

Sempre que o programa executar alguma destas opções, ele deve imprimir a lista de alunos na ordem do vetor e imprimir logo em seguida o menu antes de pedir por nova opção. Além disso, o programa deve usar apenas de uma rotina de ordenação para ordenar os alunos. Os diferentes tipos de ordenação são definidos usando funções de comparação específicas para cada tipo de ordenação que devem ser passadas como parâmetros da rotina de ordenação.