



Normas Gerais do Projeto
Nomenclatura na Implementação

Página 1 de 14

NGP. 04

Processo Seletivo

Projeto Integrado 5º Semestre

Nome	Matrícula
Denise Aparecida de Lima	04
Evaldo Aurelio Alves de Lavos	31
Mara Cristina Leite	19
Rogério Luiz Alves Lopes	50
Cícero Marcelo da Silva	52
Cláudio Alves de Lima	25
Clóvis Alves de Lima	02
Robson de Sousa Martins	48

Revisão	Alterações	Originador	Verificador	Data
0	Liberação de Documento	Robson	Denise	18/03/2002
A	Inclusão das normas para nomenclatura de componentes (pág.6) e identificação do código-fonte (pág.9)	Robson	Evaldo	22/03/2002
B				
C				
D				
E				
F				
G				
H				
I				
J				
L				

SUMÁRIO

1	OBJETIVO	3
2	DOCUMENTOS DE REFERÊNCIA	3
3	PADRÕES PARA A IMPLEMENTAÇÃO DO APLICATIVO	3
3.1	NOMENCLATURA DE ARQUIVOS	4
3.2	NOMENCLATURA DOS COMPONENTES	5
3.3	ORGANIZAÇÃO DO CÓDIGO-FONTE	8
3.3.1	<i>Regras de indentação</i>	<i>9</i>
3.4	VERSÃO DO CÓDIGO-FONTE	14

1 Objetivo

Este documento tem por objetivo estabelecer as normas para a nomenclatura de arquivos, componentes e itens relativos à implementação do Projeto Sistema Seletivo, proporcionando um padrão que permitirá uma melhor manutenibilidade e compreensão do sistema.

2 Documentos de Referência

Não Aplicável.

3 Padrões para a Implementação do Aplicativo

O aplicativo do Projeto Processo Seletivo será implementado através do ambiente de programação Borland Delphi™ versão 6.0.

A seguir estão os padrões para a nomenclatura dos arquivos e componentes utilizados no ambiente de programação.

Como este documento foi baseado no Delphi, vários das regras descritas a seguir podem não ser aplicáveis à linguagem utilizada. Onde houver correspondência, no entanto, as normas de nomeação e indentação devem ser seguidas.

Padronização de codificação é um tópico extremamente polêmico, já que está diretamente relacionado com o estilo pessoal de programação de cada indivíduo. Mudanças no estilo geralmente não são bem-vindas, pois o programador já está acostumado com o seu próprio estilo e sempre acha que o seu é melhor. No entanto, um estilo padrão de codificação é particularmente útil a partir do momento em que os fontes começam a sofrer manutenção por outros programadores que não escreveram o código original, reduzindo o tempo necessário para alterações e minimizando a ocorrência de efeitos cascata e/ou colaterais (conserta uma rotina, prejudica outra).

Sugerimos aqui um padrão com certa liberdade para o estilo pessoal, mas regido por regras básicas e simples. Um código fonte deve ser bem organizado, conter comentários e evitar funções e procedimentos muito longos. Uma indentação adequada mantém o código bem organizado, assim como procedimentos curtos que utilizem as estratégias “dividir para conquistar” e “se todas as partes funcionam, o todo funciona”. Estas estratégias serão discutidas ao longo do documento. Todos os exemplos serão mostrados em Pascal e com seu equivalente em C.

3.1 Nomenclatura de arquivos

Os arquivos-fonte gerados pelo Borland Delphi™ deverão seguir o padrão descrito:

- Units

As units (.pas) terão seu nome composto da seguinte forma:

u_ParaQueServeaUnit.pas

onde:

ParaQueServeaUnit é uma descrição que representa o uso da unit, sem uso de espaços, hífens ou *underlines*. Por exemplo:

u_CadastroAlunos.pas – é uma unit que contém código relativo ao form de cadastro de alunos.

Os arquivos que contém os forms (.dfm) terão conseqüentemente os mesmos nomes dos arquivos das units correspondentes, porém com a extensão .dfm.

- Outros arquivos

Se existirem outros arquivos, como arquivos de configuração, etc. estes terão o nome de arquivo e extensão que melhor convém ao projeto.

3.2 Nomenclatura dos componentes

Os componentes usados na implementação do projeto devem seguir o seguinte padrão:

xxx_yyyy

onde:

xxx é o tipo de componente (usar tabela a seguir).

yyyy é um nome, sem espaços, hífen e *underlines*, único, que representa o significado do componente.

Por exemplo:

edt_Senha : um componente TEdit que serve para a digitação de uma senha.

edt_Telefone : um componente TDBEdit que representa o campo Telefone em uma tabela.

Imgl_32 : um componente TImageList que contém ícones de 32x32 pixels.

Para os forms, deve ser seguido o padrão:

frm_ParaQueServeaUnit

Observe que ParaQueServeaUnit é a mesma descrição usada no nome do arquivo da unit correspondente a esse form.

Pref	Tipo de Objeto	Pref	Tipo de Objeto
bar	ProgressBar, StatusBar, TrackBar e outros tipos de bar	nav	DBNavigator
bat	BatchMove	nbn	Notebook, TabbedNotebook
btn	Todos os tipos de Buttons que não sejam spins	nnt	NNTP (Internet Network Newsgroup Access)
bvl	Bevel	ole	OLEContainer
cal	Calendar	out	Outline
cbx	ComboBox, DBComboBox	pbx	PaintBox
chk	CheckBox, DBCheckBox	pge	PageControl
db	Database	pnl	Todos os tipos de Panel
dcc	DDEClientConv	pop	POP (Internet Post Office Protocol to receive E-Mail)
dci	DDEClientItem	qry	Todos os tipos de Query
dir	DirectoryOutline	rad	RadioButton
dlg	Dialogs (Open, Save, Font, Color, etc...)	rgp	RadioGroup, DBRadioGroup
drv	DriveComboBox	rpt	Report
dsc	DDEServerConv	scb	Scrollbar
dsi	DDEServerItem	sbx	Scrollbar
dtm	DataModule	shp	Shape
ds	DataSource	sht	VCFormulaOne
edt	Todos os tipos de Edit Box	smt	SMTP (Internet Simple Mail Transport)

Pref	Tipo de Objeto	Pref	Tipo de Objeto
cbx	Todos os tipos de Combo Box que não sejam lookups	spl	VCSpeller
frm	Form	spn	SpinButton, UpDown
fme	Frame	ssn	Session
ftp	FTP (Internet File Transfer Protocol)	stp	StoredProc
gge	Gauge	tab	TabControl, Tabset
grd	Todos os grids	tsh	TabSheet
grf	ChartfX, VCFirstImpression, GraphicsServer	tbl	Todos os tipos de Table
grp	GroupBox	tcp	TCP (Internet Data Exchange ... like a telephone)
hdr	HeaderControl, Header	tmr	Timer
hot	Hotkey	tvw	TreeView
htm	HTML (Internet Web Browser)	udp	UDP (Internet Data Broadcast ... like a radio)
htt	HTTP (Send, Receive or Search HTML Documents)	ups	UpdateSQL
ibe	IBEventAlerter	mmo	Memo, DBMemo, RichEdit
img	Todos os tipos de Image	mni	MenuItem
imgl	Todos os tipos de ImageList	mnu	MainMenu, PopupMenu
lbl	Todos os tipos de Labels	imgr	TfclMager
lbx	Todos os tipos de List Box	btng	TfcButtonGroup
lkp	Todos os tipos de Lookups Combo	dc	TTXDatasetControl

Pref	Tipo de Objeto	Pref	Tipo de Objeto
lvw	ListView	dt	Todos os tipos de DateTimePicker
Med	MediaPlayer		

3.3 Organização do código-fonte

Para melhorar o entendimento do código-fonte, é necessário a inclusão de comentários, que podem ser redigidos livremente pelo programador.

A indentação do código deverá seguir as normas especificadas no próximo item.

As units que correspondem a forms deverão conter código relativo ao próprio form. Processamentos especiais ou comuns a mais de uma unit deverão estar alocados em uma unit separada.

Deverá ser utilizada para a codificação a OOP, evitando-se uso de estruturas, variáveis ou funções repetidas e não-reutilizáveis.

Para possibilitar um melhor desempenho da aplicação (uso de memória e recursos de sistemas otimizado) deve-se utilizar o método de criação dinâmica de objetos, incluindo forms, matrizes e outras estruturas complexas, como as consultas ao BD e os relatórios.

3.3.1 Regras de indentação

3.3.1.1 1ª regra de indentação

Sempre utilizar uma única indentação para cada novo nível de comandos.

Indentação é a organização dada ao código para indicar grupos de comando relacionados. Fontes sem indentação são muito difíceis de manipular, pois não se tem um feedback visual da estrutura lógica dos comandos a serem executados. Já fontes com indentação exagerada deslocam o código muito para a direita, deixando grandes espaços em branco e dificultando da mesma forma a visualização. Utilizaremos para cada novo nível de comando uma única indentação de 2 ou 3 espaços. Exemplos:

Exemplo 1 (Incorreto)	Exemplo 2 (Correto)	Exemplo 3 (Correto)
<pre>Max:= 10; i:= 0; while (i < Max) do begin if (x < Max) then begin x:= i*2; y:= y+x; end; inc(i); end;</pre>	<pre>Max:= 10; i:= 0; while (i < Max) do begin if (x < Max) then begin x:= i*2; y:= y+x; end; inc(i); end;</pre>	<pre>Max:= 10; i:= 0; while (i < Max) do begin if (x < Max) then begin x:= i*2; y:= y+x; end; inc(i); end;</pre>
<pre>Max = 10; i = 0; while (i < Max) { if (x < Max) { x = i*2; y += x; } i++; }</pre>	<pre>Max = 10; i = 0; while (i < Max) { if (x < Max) { x = i*2; y += x; } i++; }</pre>	<pre>Max = 10; i = 0; while (i < Max) { if (x < Max) { x = i*2; y += x; } i++; }</pre>

O primeiro exemplo está errado porque para cada novo nível de comando foram utilizadas duas indentações: uma para o begin e outra para os comandos. O segundo e o terceiro exemplos estão corretos porque utilizam apenas uma indentação para cada nível de comando.

Existe uma pequena diferença de estilo entre os dois últimos exemplos que é permitida. Alguns programadores preferem ver o begin e o end (ou o { e } do C) como pares, o que evita o erro de ter um end sem begin ou vice-versa. É o que acontece no exemplo 2. Outros preferem ver rapidamente qual bloco de comandos o end está encerrando (o último end encerra o while, o penúltimo o if, etc) ou gostam de economizar as linhas gastas pelo begin. É o que acontece no exemplo 3.

3.3.1.2 2ª regra de indentação

Se um bloco de comandos for composto por um único comando, evite begin...end e coloque o comando na mesma linha sem indentação ou na linha seguinte com indentação.

As vezes um bloco de comandos é composto por apenas um comando. Nestes casos é mais claro deixar o bloco em uma mesma linha e não usar begin...end nem indentação. No entanto, se a linha ficar muito longa e dificultar a visualização do código, deve-se quebrar a linha e utilizar a regra de indentação (veja o exemplo 4). Os exemplos abaixo estão todos corretos, apenas mostram os diferentes estilos possíveis de acordo com a regra e como a visualização do código é facilitada ou prejudicada.

Exemplo 1

```
Repeat
  inc(i);
until (i >= 10);
```

ou...

```
repeat inc(i) until (i >= 10);
```

Exemplo 2

```
While (i < 10) do begin
  inc(i);
end;
```

ou...

```
while (i < 10) do inc(i);
```

Exemplo 3

```
if (x > 10) then begin
  x:= 10;
end
else begin
  x:= x+1;
end;
```

ou...

```
if (x > 10) then x:= 10
else x:= x+1;
```

Exemplo 4

```
if (x > 10) then begin
  NomeDeRotinaMuitoLongo()
end
else begin
  OutroNomeDeRotinaMuitoLongo()
end;
```

ou...

```
if (x > 10) then NomeDeRotinaMuitoLongo()
else OutroNomeDeRotinaMuitoLongo();
```

ou...

```
if (x > 10) then
  NomeDeRotinaMuitoLongo()
else
  OutroNomeDeRotinaMuitoLongo();
```

3.3.1.3 3º regra de indentação

Alinhe sempre um else com seu if correspondente. Um comando else pertence ao mesmo nível do if, e não deve sofrer indentação relativa ao if.

Muitos programadores consideram o else como a contrapartida do then. Essa confusão ocorre somente em Pascal, que usa a palavra then após a condição (if <condição> then <faça alguma coisa>). Outras linguagens não usam a palavra then. Na verdade, o else é a contrapartida do if, e portanto deve estar alinhado com o mesmo.

Os exemplos abaixo mostram códigos incorretamente indentados e já encontrados em várias implementações:

Exemplo 1 (incorreto)

```
if (s = 'sim')
then begin
    Rotina1();
    Rotina2();
    Rotina3();
end
else Rotina4();
```

Exemplo 1 corrigido

```
if (s = 'sim') then
begin
    Rotina1();
    Rotina2();
    Rotina3();
end
else Rotina4();
```

Exemplo 2 (incorreto)

```
if (s = 'sim') then begin
    Rotina1();
    Rotina2();
    Rotina3();
end
else begin
    Rotina4();
    Rotina5();
end;
```

Exemplo 2 corrigido

```
if (s = 'sim') then begin
    Rotina1();
    Rotina2();
    Rotina3();
end
else begin
    Rotina4();
    Rotina5();
end;
```

3.3.1.4 4º regra de indentação

Em if's aninhados, se o bloco de comandos após o else for composto por um único if, considere o par else if como um único comando que deve estar alinhado com o if (ou com outro else if) correspondente.

O comando if..else pode causar confusão na indentação quando aparece aninhado (comandos if..else dentro de outros). Um if aninhado se assemelha muito a um case do Pascal ou um switch do C. Assim, cada comando else seguido de um único comando if deve ser tratado como se fosse um único comando (algumas linguagens até possuem o comando elsif para facilitar). Um comando else if deve estar alinhado com seu if ou else if correspondente, e segue todas as outras regras. Observe os exemplos a seguir.

Exemplo 1 (incorreto)

```
if (s = 'um') then Rotina1()
else begin
  if (s = 'dois') then Rotina2()
  else begin
    if (s = 'tres') then Rotina3()
    else begin
      if (s = 'quatro') then Rotina4()
      else Rotina5();
    end;
  end;
end;
end;
```

Exemplo 1 corrigido

```
if (s = 'um') then Rotina1()
else if (s = 'dois') then Rotina2()
else if (s = 'tres') then Rotina3()
else if (s = 'quatro') then Rotina4()
else Rotina5();
```

Exemplo 2 (incorreto)

```
if (s = 'um') then begin
  Rotina11();
  Rotina12();
end
else begin
  if (s = 'dois') then begin
    Rotina21();
    Rotina22();
  end;
  else begin
    if (s = 'tres') then begin
      Rotina31();
      Rotina32();
    end;
    else begin
      Rotina41();
      Rotina42();
    end;
  end;
end;
end;
```

Exemplo 2 corrigido

```
if (s = 'um') then begin
  Rotina11();
  Rotina12();
end
else if (s = 'dois') then begin
  Rotina21();
  Rotina22();
end
else if (s = 'tres') then begin
  Rotina31();
  Rotina32();
end
else begin
  Rotina41();
  Rotina42();
end;
```

3.4 Versão do código-fonte

Para um melhor controle do código-fonte, ele deverá ser marcado com uma versão, sendo 0.0.0.0 a primeira versão (em tempo de implementação) e 1.0.0.0 a primeira versão final ao cliente.