

Henrique Loureiro

C++

GUIA MODERNO DE PROGRAMAÇÃO

- Desenho de interfaces gráficas (Gtkmm)
- Manipulação de bases de dados (MySQL)
- Integração com XML (TinyXML)

Código compatível com



Linux



Windows



EDIÇÃO

FCA – Editora de Informática, Lda.
Av. Praia da Vitória, 14 A – 1000-247 Lisboa
Tel: +351 213 511 448
fca@fca.pt
www.fca.pt

DISTRIBUIÇÃO

Lidel – Edições Técnicas, Lda.
Rua D. Estefânia, 183, R/C Dto. – 1049-057 Lisboa
Tel: +351 213 511 448
lidel@lidel.pt
www.lidel.pt

LIVRARIA

Av. Praia da Vitória, 14 A – 1000-247 Lisboa
Tel: +351 213 511 448 * Fax: +351 213 522 684
livraria@lidel.pt

Copyright © 2019, FCA – Editora de Informática, Lda.

® Marca registada

ISBN edição impressa: 978-972-722-904-8

1.ª edição impressa: setembro 2019

Impressão e acabamento: Tipografia Lousanense, Lda. - Lousã

Depósito Legal n.º 460731/19

Capa: *Look-Ahead* – José Manuel Ferrão

Todos os nossos livros passam por um rigoroso controlo de qualidade, no entanto aconselhamos a consulta periódica do nosso *site* (www.fca.pt) para fazer o *download* de eventuais correções.

Não nos responsabilizamos por desatualizações das hiperligações presentes nesta obra, que foram verificadas à data de publicação da mesma.

Os nomes comerciais referenciados neste livro têm patente registada.



Reservados todos os direitos. Esta publicação não pode ser reproduzida, nem transmitida, no todo ou em parte, por qualquer processo eletrónico, mecânico, fotocópia, digitalização, gravação, sistema de armazenamento e disponibilização de informação, sítio Web, blogue ou outros, sem prévia autorização escrita da Editora, exceto o permitido pelo CDADC, em termos de cópia privada pela AGE COP – Associação para a Gestão da Cópia Privada, através do pagamento das respetivas taxas.

ÍNDICE GERAL

PRÓLOGO	XI
COMO UTILIZAR ESTE LIVRO.....	XIII
PARTE I - PRINCÍPIOS GERAIS	1
1. INTRODUÇÃO.....	3
1.1 Convenções de código.....	3
1.1.1 Regras de nomenclatura	3
1.1.2 Comentários	4
1.1.3 Indentação	4
1.2 Estrutura básica	5
1.3 Variáveis	6
1.3.1 Atribuição de valores a variáveis.....	7
1.3.2 Utilização de valores contidos em variáveis	7
1.4 Constantes	8
1.5 Saída (<i>output</i>) e entrada (<i>input</i>) de dados	8
1.6 Fases de desenvolvimento	9
1.6.1 Fase I: Declarações.....	10
1.6.2 Fase II: Entrada	10
1.6.3 Fase III: Cálculo	10
1.6.4 Fase IV: Saída	10
1.6.5 Fase V: Implementação	10
1.7 Exercícios propostos.....	11
2. TIPOS DE DADOS.....	13
2.1 Tipos de dados primitivos	13
2.1.1 Numéricos	13
2.1.2 Lógicos	15
2.1.3 Carateres.....	15
2.2 Tipos de dados compostos.....	16
2.2.1 Vetores	16
2.2.2 Matrizes	17
2.2.3 Cadeias de carateres alfanuméricos	18
2.2.4 Estruturas de dados	19
2.3 Conversão entre tipos de dados.....	20
2.3.1 Conversão implícita	20
2.3.2 Conversão explícita	20
2.4 Exercícios propostos	21
3. OPERADORES.....	23

3.1	Operações de atribuição.....	23
3.2	Operações aritméticas	23
3.3	Operações relacionais.....	25
3.4	Operações lógicas.....	26
3.5	Exercícios propostos.....	27
4.	ESTRUTURAS DE CONTROLO.....	29
4.1	Decisões.....	29
4.1.1	If.....	29
4.1.2	If else	30
4.1.3	If else if.....	31
4.1.4	Switch.....	32
4.2	Repetições	33
4.2.1	While	33
4.2.2	Do while.....	34
4.2.3	For.....	34
4.2.4	Ciclos infinitos e quebra de ciclos	35
4.3	Exercícios propostos	35
5.	FUNÇÕES.....	37
5.1	Tipos de funções	37
5.1.1	Funções do tipo A.....	37
5.1.2	Funções do tipo B.....	38
5.1.3	Funções do tipo C.....	39
5.1.4	Funções do tipo D.....	39
5.2	Ciclo de vida das variáveis	40
5.3	Protótipos.....	41
5.4	Exercícios propostos	42
PARTE II -	PROGRAMAÇÃO ORIENTADA A OBJETOS	43
6.	CLASSES.....	45
6.1	Conceitos gerais	45
6.2	Objetos.....	46
6.2.1	Atributos.....	46
6.2.2	Métodos	47
6.2.3	Instâncias	48
6.2.4	Construtores.....	49
6.2.5	Destrutores	50
6.3	Protótipos.....	51
6.4	Exercícios propostos	52
7.	HERANÇA	53
7.1	Herança única.....	53
7.2	Herança múltipla	56
7.3	Hierarquia.....	57

7.4 Exercícios propostos	60
8. POLIMORFISMO.....	61
8.1 Ponteiros	61
8.1.1 Ponteiros para variáveis	61
8.1.2 Ponteiros para objetos.....	62
8.1.3 This.....	63
8.2 Métodos virtuais	65
8.3 Exercícios propostos	66
PARTE III - BIBLIOTECAS-PADRÃO	67
9. CMATH	69
9.1 Funções trigonométricas	69
9.2 Funções exponenciais	70
9.3 Funções de potenciação.....	70
9.4 Funções de arredondamento	71
9.5 Exercícios propostos	72
10. CSTRING.....	73
10.1 Medição.....	73
10.2 Comparação.....	74
10.3 Extração.....	75
10.4 Substituição.....	76
10.5 Pesquisa.....	76
10.6 Exercícios propostos	77
11. CTIME	79
11.1 Estrutura tm.....	79
11.2 Data local.....	80
11.3 Formatação	81
11.4 Cálculos.....	82
11.4.1 Período entre datas.....	82
11.4.2 Adição e subtração de tempo.....	83
11.5 Exercícios propostos	84
12. CSTDIO E CSTDLIB.....	85
12.1 Formatação de cadeias de caracteres alfanuméricos.....	85
12.1.1 Função printf.....	85
12.1.2 Função vprintf.....	86
12.2 Conversão de cadeias de caracteres alfanuméricos.....	87
12.3 Geração de números aleatórios	88
12.4 Comandos de sistema.....	88
12.5 Exercícios propostos	89
PARTE IV - ACESSO A DADOS.....	91

13. SISTEMA DE FICHEIROS.....	93
13.1 Verificação de itens.....	93
13.2 Criação de diretórios	94
13.3 Eliminação de itens.....	95
13.4 Renomeação de itens	95
13.5 Exercícios propostos	96
14. FICHEIROS DE TEXTO.....	97
14.1 Classes fundamentais	97
14.2 Escrita	97
14.2.1 Método substitutivo	98
14.2.2 Método aditivo.....	98
14.3 Leitura	99
14.3.1 Método indiferenciado	99
14.3.2 Método diferenciado por linha	100
14.3.3 Método diferenciado por valor.....	101
14.4 Edição e cópia.....	101
14.5 Exercícios propostos	102
15. INTEGRAÇÃO COM XML.....	105
15.1 Sintaxe XML.....	105
15.1.1 Declaração XML.....	105
15.1.2 Elementos e atributos.....	105
15.1.3 Comentários	106
15.2 Criação de um ficheiro XML.....	107
15.3 Leitura e navegação	107
15.4 Inclusão de elementos	109
15.4.1 1.º nível.....	109
15.4.2 2.º nível.....	110
15.5 Modificação de atributos.....	112
15.6 Exclusão de elementos	113
15.7 Exercícios propostos	114
16. ACESSO AO MYSQL	117
16.1 Apresentação de um caso real.....	117
16.1.1 Acesso ao servidor.....	117
16.1.2 Criação da base de dados	118
16.1.3 Criação das tabelas de dados	119
16.1.4 Inserção de registos de teste.....	120
16.2 Obtenção de listagens.....	121
16.2.1 Consultas de seleção	121
16.2.2 Ligação ao servidor	122
16.2.3 Obtenção de toda a informação numa tabela	123
16.2.4 Ordenação de listagens.....	124
16.2.5 Aplicação de critérios.....	124
16.2.6 Pesquisa de informação	124

16.2.7 Contagem de registos.....	125
16.2.8 Agrupamento da informação.....	126
16.2.9 Aplicação de critérios em dados agrupados	127
16.2.10 Relação entre tabelas.....	127
16.3 Manipulação de dados	128
16.3.1 Inserção de registos	128
16.3.2 Atualização de registos.....	129
16.3.3 Exclusão de registos	130
16.4 Exercícios propostos	131
PARTE V - INTERFACES GRÁFICAS	133
17. USABILIDADE	135
17.1 As 10 heurísticas de Nielsen	135
17.1.1 <i>Feedback</i>	136
17.1.2 Linguística	136
17.1.3 Interrupção	136
17.1.4 Consistência	137
17.1.5 Prevenção	137
17.1.6 Memorização.....	137
17.1.7 <i>Shortcuts</i>	137
17.1.8 Clareza	137
17.1.9 Descomplicar.....	138
17.1.10 Documentação	138
17.2 <i>Design</i> de aplicações <i>desktop</i>	138
17.2.1 Modos de ecrã.....	138
17.2.2 Tipos de janelas.....	139
17.2.3 Caixas de controlo	139
17.2.4 Redimensionamento e mobilidade.....	139
17.2.5 Barras de estado.....	140
17.2.6 <i>Zoom</i>	140
17.2.7 Barras de deslocamento	140
17.2.8 Área de notificação.....	141
17.2.9 Menus e barras de ferramentas.....	141
17.2.10 Barras de progresso.....	141
17.2.11 Psicologia das cores.....	141
17.2.12 Apresentação de texto.....	143
17.2.13 Seleção de opções	143
18. GTK ELEMENTAR.....	145
18.1 Janelas.....	145
18.2 <i>Layouts</i>	147
18.2.1 <i>Alignment</i>	147
18.2.2 <i>Fixed</i>	148
18.2.3 <i>VBox</i> e <i>HBox</i>	150
18.2.4 <i>Grid</i>	151

18.3 Exercícios propostos	153
19. INTERAÇÃO	155
19.1 Eventos	155
19.2 Barras de menus	156
19.3 Barras de ferramentas	158
19.4 Barras de estado	160
19.5 Exercícios propostos	161
20. DIÁLOGO	163
20.1 Mensagens	163
20.2 Tipos de letra	165
20.3 Cores	167
20.4 Ficheiros	169
20.5 Exercícios propostos	171
PARTE VI - PROJETOS	173
21. PROJETO I: CALCULADORA	175
21.1 <i>Layout</i>	175
21.2 Definição da classe	175
21.3 Construtor	177
21.4 Eventos de clique dos botões de dígitos	179
21.5 Eventos de clique dos botões de operações	179
21.6 Cálculo	180
21.7 <i>Reset</i>	181
21.8 Exercícios propostos	182
22. PROJETO II: VIDEO POKER	183
22.1 <i>Layout</i>	183
22.2 Regras	183
22.3 Definição da classe	185
22.4 Construtor	186
22.5 Inicializações	188
22.6 Atualização de créditos	188
22.7 Reposição de baralho	189
22.8 <i>Deal</i>	190
22.9 Primeiras cartas	191
22.10 Seleção de cartas	192
22.11 <i>Draw</i>	192
22.12 Novas cartas	192
22.13 Saber prémio	193
22.14 Exercícios propostos	198
23. PROJETO III: HOTÉIS	199
23.1 <i>Layout</i>	199

23.2	Configuração do modelo de dados.....	200
23.3	Definição da classe.....	200
23.4	Construtor.....	201
23.5	Exercícios propostos	202
24.	PROJETO IV: COTAÇÕES	203
24.1	<i>Layout</i>	203
24.2	Definição da classe.....	204
24.3	Construtor.....	204
24.4	Exercícios propostos	207
25.	PROJETO V: SONDAAGEM	209
25.1	Objetivo	209
25.2	Desenho da base de dados.....	210
25.3	<i>Layout</i>	211
25.4	Definição da classe.....	212
25.5	Construtor.....	214
25.6	Atualização do contador.....	217
25.7	Leitura de distritos.....	218
25.8	Limpeza do formulário de recolha de dados	219
25.9	Permuta de ativação empregados/desempregados.....	219
25.10	Inserção de registo e mensagem	220
25.11	Atualização da taxa de desemprego.....	223
25.12	Exercícios propostos	224
	GLOSSÁRIO DE TERMOS – PORTUGUÊS EUROPEU/PORTUGUÊS DO BRASIL.....	225
	ÍNDICE REMISSIVO.....	227

COMO UTILIZAR ESTE LIVRO

Com o objetivo de facilitar uma aprendizagem rápida e eficaz, tomamos a liberdade de sugerir um conjunto de indicações às quais o leitor deverá dar especial atenção, de forma a tirar o maior proveito deste livro.

CONTEÚDOS PROGRAMÁTICOS

Os diferentes temas foram encadeados pedagógica e naturalmente, pelo que não é, de todo, recomendado que progrida sem a compreensão das matérias abordadas em capítulos anteriores. A tabela que se segue apresenta um resumo dos principais conteúdos tratados nesta obra.

PARTE		CAPÍTULO		PRINCIPAIS TEMAS/ASSUNTOS
I	Princípios Gerais	1	Introdução	Convenções de código
				Primeiras aplicações C++
				Variáveis e constantes
				Fluxo de dados
		2	Tipos de Dados	Tipos de dados primitivos
				Tipos de dados compostos
				Conversões
		3	Operadores	Operadores de atribuição
				Operadores aritméticos
				Operadores relacionais
				Operadores lógicos
		4	Estruturas de Controlo	Decisões
				Repetições
		5	Funções	Tipos de funções
				Argumentação
				Retorno
				Protótipos
II	Programação Orientada a Objetos	6	Classes	Noção de objeto
				Atributos
				Métodos
				Instâncias
				Construtores

PARTE		CAPÍTULO		PRINCIPAIS TEMAS/ASSUNTOS
II	Programação Orientada a Objetos			Destrutores
				Protótipos
		7	Herança	Herança única
				Herança múltipla
				Considerações hierárquicas
		8	Polimorfismo	Ponteiros
III	Bibliotecas-Padrão			Métodos virtuais
		9	cmath	Funções trigonométricas
				Funções exponenciais
				Funções de potenciação
				Funções de arredondamento
		10	cstring	Medição de caracteres
				Comparação de texto
				Extração de texto
				Substituição de texto
				Pesquisa de caracteres
		11	ctime	Estrutura de dados temporais
				Obtenção de datas locais
				Formatação de datas
				Cálculos entre datas
		12	cstdio e cstdlib	Formatação de texto
				Conversão de texto
				Números aleatórios
				Comandos de sistema
IV	Acesso a Dados	13	Sistema de Ficheiros	Verificação de itens (ficheiros e diretórios)
				Criação de itens (diretórios)
				Eliminação de itens (ficheiros e diretórios)
				Renomeação de itens (ficheiros e diretórios)
		14	Ficheiros de Texto	Escrita
				Leitura
				Edição
				Cópia
		15	Integração com XML	Considerações sintáticas
				Criação de documentos
				Leitura
				Navegação
		16	Acesso ao MySQL	Escrita
				Desenho de bases de dados
				Ligação ao servidor

PARTE		CAPÍTULO		PRINCIPAIS TEMAS/ASSUNTOS
V	Interfaces Gráficas			Obtenção de listagens
				Manipulação de dados
		17	Usabilidade	Heurísticas de Nielsen
				Considerações a ter ao nível do desenvolvimento de aplicações <i>desktop</i>
		18	GTK Elementar	Criação de aplicações gráficas
				Implementação de janelas
				<i>Layouts</i>
		19	Interação	Sinais e eventos
				Barras de menus
				Barras de ferramentas
				Barras de estado
		20	Diálogo	Emissão de mensagens
				Escolha de tipos de letra
				Escolha de cores
				Seleção de itens (diretórios e ficheiros)
VI	Projetos	21	Projeto I: Calculadora	Calculadora de bolso virtual
		22	Projeto II: <i>Video Poker</i>	Jogo de <i>Video Poker</i>
		23	Projeto III: Hotéis	Pesquisa de hotéis (informação contida num ficheiro de texto)
		24	Projeto IV: Cotações	Cotações monetárias (uso de um ficheiro XML <i>online</i>)
		25	Projeto V: Sondagem	Sondagem de avaliação da taxa de desemprego (interação com uma base de dados MySQL)

SOFTWARE UTILIZADO

Todas as listagens de código compilável foram testados em **Code::Blocks** e são compatíveis com os sistemas operativos **Linux** e **Windows**, não se prevendo que venham a ocorrer alterações significativas entre as atuais versões e eventuais *updates*.

Consulte o **Guia de Instalação**, disponível na página do livro em www.fca.pt, para obter informações detalhadas acerca dos componentes de *software* complementares e dos respetivos processos de instalação e configuração.

ANEXO E MATERIAIS COMPLEMENTARES

O anexo e materiais complementares mencionados ao longo da obra encontram-se disponíveis para *download* na página do livro em www.fca.pt.

GLOSSÁRIO – PORTUGUÊS EUROPEU/PORTUGUÊS DO BRASIL

Para facilitar a interpretação do texto por parte dos leitores brasileiros, é incluído um glossário de termos correspondentes que são utilizados pelo autor na explicação das matérias. Este glossário pode ser consultado no final do livro.

CONTACTAR O AUTOR

No sentido de proporcionar aos leitores uma ferramenta complementar ao conhecimento transmitido através desta obra, o autor disponibiliza-se a ser contactado para auxiliar os leitores a ultrapassarem eventuais dificuldades que sintam na interpretação das matérias abordadas e/ou na resolução dos variados exercícios apresentados ao longo deste livro.

Assim, e com o objetivo de facilitar a resposta por parte do autor, agradecemos que considere os pontos que se seguem aquando do seu contacto:

- Identifique claramente a obra (*C++ – Guia Moderno de Programação*);
- Verifique se, porventura, existirá uma errata disponível na página do livro em www.fca.pt;
- Refira os números do capítulo e da página, caso o seu pedido de informação esteja diretamente relacionado com o conteúdo do livro;
- Indique a versão e a edição do seu sistema operativo (por exemplo, **Ubuntu 16.04 LTS, Windows 10 Pro**);
- Especifique a versão do **Code::Blocks** que está a utilizar (por exemplo: **Code::Blocks 17.12, Code::Blocks 16.01**);
- Se a sua questão remeter para o desenvolvimento de aplicações baseadas em **MySQL**, identifique a versão em uso (por exemplo, **MySQL 8.0.17**);
- Se determinante, proceda ao envio do(s) ficheiro(s) referente(s) ao(s) seu(s) projeto(s);
- Inclua capturas de ecrã com as mensagens de erro, se for o caso.

Todos os pedidos de esclarecimento, assim como eventuais sugestões, poderão ser enviados diretamente para o endereço do autor: loureirohenrique@hotmail.com.

1

INTRODUÇÃO

Este capítulo introduz os conceitos triviais da linguagem C++, tais como o uso elementar de variáveis e constantes, a realização de cálculos simples e a comunicação básica com o utilizador. Após a leitura e o entendimento da matéria aqui tratada, o leitor ficará apto a solucionar algoritmos de fácil interpretação e montagem.

1.1 CONVENÇÕES DE CÓDIGO

Esta secção descreve as normas relativas à apresentação de código-fonte para que o leitor interprete de forma adequada os exemplos apresentados ao longo do livro.

1.1.1 REGRAS DE NOMENCLATURA

Logicamente, teremos de criar identificadores próprios para os elementos que vamos introduzindo ao longo do processo de desenvolvimento das nossas aplicações, tais como nomes de funções, parâmetros e variáveis, que só nós conhecemos e utilizamos e que não estão naturalmente presentes na linguagem. Assim, é de extrema importância seguir um padrão no que concerne à escrita de código-fonte, pois em grande parte dos projetos estão envolvidos vários colaboradores, e esta uniformização facilita bastante a leitura e o posterior desenrolamento.

No processo de atribuição de nomes em C++, tenha em consideração os seguintes aspetos:

- A linguagem é *case-sensitive*, ou seja, sensível a maiúsculas e minúsculas;
- O primeiro carácter deve ser uma letra minúscula;
- Não podem ser usados acentos gráficos;
- Não podem existir espaços;
- Não podem existir caracteres especiais, tais como ponto final (.), vírgula (,), ponto e vírgula (;), hífen (-), arroba (@), símbolo monetário (\$), cardinal (#) e o E comercial (&);
- Os nomes são únicos no módulo de código em que são definidos.



É comum alternar-se entre maiúsculas e minúsculas para “separar” duas palavras (por exemplo, importarDados, memoriaLivre, imprimirDocumento). De salientar, ainda, que os nomes devem conter poucos caracteres, sem prejuízo da sua lucidez e do papel que representam e desempenham.

1.1.2 COMENTÁRIOS

Os comentários são particularmente úteis para documentar código e ignorados aquando do momento da compilação. Há dois tipos de comentários em C++: de **linha única** e de **linhas múltiplas**.

Quando empregamos duas barras (//), estamos a considerar que tudo o que estiver à sua direita até ao final da linha é um comentário de linha única; quando incluímos texto entre os caracteres /* e */, estamos a informar o compilador de que tudo o que estiver entre eles deve ser considerado um comentário de linhas múltiplas. No próximo exemplo, todo o texto a cinzento é considerado comentário:

```
int main() //Função main
{
    //O seu código aqui...
}
/* Listagem criada por Henrique Loureiro
@ setembro 2019 */
```

1.1.3 INDENTAÇÃO

A indentação é uma técnica muito conhecida que consiste em tornar mais fáceis a leitura e a eventual modificação do código-fonte, permitindo ainda a disposição de blocos de instruções segundo uma hierarquia lógica e estruturante. Cada bloco pode conter um número ilimitado de sub-blocos, dependendo das estruturas utilizadas (código dentro de código).

A indentação é feita com recurso a tabulações no início de cada linha, na quantidade equivalente à profundidade em que cada linha está contida:

```
int main()
{
    int resultado;
    if(valor != 0)
    {
        resultado = valor * 2;
    }
    else
    {
        resultado = 0;
    }
}
```


2

TIPOS DE DADOS

Os tipos de dados representam não só o género de informação que as variáveis e as constantes podem guardar, como também estabelecem limites mínimos e máximos dos valores suportados. À semelhança de outras linguagens de programação, a C++ possui internamente um conjunto de tipos de dados, que são a base para a criação de outros mais sofisticados e de maior versatilidade. Este capítulo apresenta uma descrição dos tipos de dados existentes em C++.

2.1 TIPOS DE DADOS PRIMITIVOS

Os tipos de dados primitivos (ou nativos) organizam-se mediante a natureza de valores que suportam, sendo distribuídos por três categorias principais (Tabela 2.1).

TIPO DE DADOS		ARMAZENA...
NUMÉRICOS	int	Números inteiros.
	float	Números reais (precisão até 7 casas decimais).
	double	Números reais (precisão até 15 casas decimais).
LÓGICOS	bool	Valores lógicos.
CARATERES	char	Carateres (ASCII ¹).

TABELA 2.1 – Tipos de dados primitivos

2.1.1 NUMÉRICOS

Dependendo de haver, ou não, necessidade de se usar **modificadores de faixa**², é possível agrupar os tipos de dados numéricos em seis subcategorias (Tabela 2.2).

¹ *American Standard Code for Information Interchange*; esquema de codificação-padrão que abrange um total de 256 carateres. Consulte as tabelas ASCII na secção A.1 do Anexo.

² Permitem alterar o intervalo de valores suportados pelas variáveis; em C++, são utilizados os modificadores `unsigned` (para limitação a números positivos) e `short` (para redução do espaço de armazenamento).

TIPO DE DADOS	TAMANHO (BYTES)	INTERVALO
unsigned short int	2	0 a 65535
short int	2	-32768 a 32767
unsigned int	4	0 a 4294967295
int	4	-2147483648 a 2147483647
float	4	1.2e-38 a 3.4e+38
double	8	2.2e-308 a 1.8e+308

TABELA 2.2 – Tipos de dados numéricos (subcategorias)



A separação entre os tipos de dados que armazenam valores inteiros e reais surge da necessidade de controlo da velocidade de processamento da informação. Por exemplo, o tipo `int` ocupa mais recursos de sistema do que o tipo `short int`, o mesmo acontecendo em relação aos tipos `double` e `float`. Assim, há que otimizar a escolha do tipo de dados que melhor se ajusta e resulta em termos de desempenho.

A Tabela 2.3 demonstra alguns exemplos de como utilizar **corretamente** variáveis das diferentes subcategorias de tipos de dados numéricos.

DECLARAÇÃO	ATRIBUIÇÃO CORRETA
<code>unsigned short int valorUSI;</code>	<code>valorUSI = 0;</code> <code>valorUSI = 6;</code>
<code>short int valorSI;</code>	<code>valorSI = -45;</code> <code>valorSI = 0;</code> <code>valorSI = 20000;</code>
<code>unsigned int valorUI;</code>	<code>valorUI = 0;</code> <code>valorUI = 6788587;</code>
<code>int valorI;</code>	<code>valorI = -76854;</code> <code>valorI = 0;</code> <code>valorI = 2000000;</code>
<code>float valorF;</code>	<code>valorF = -45098;</code> <code>valorF = -4.6;</code> <code>valorF = 0;</code> <code>valorF = 18;</code> <code>valorF = 20.002;</code>
<code>double valorD;</code>	<code>valorD = -7.700005455;</code> <code>valorD = -8900005455;</code> <code>valorD = 0;</code> <code>valorD = 1000000000;</code> <code>valor = 86385.66;</code>

TABELA 2.3 – Exemplos de utilização correta de tipos de dados numéricos

Já a Tabela 2.4 exhibe alguns exemplos de utilização **incorreta** de variáveis das diferentes subcategorias de tipos de dados numéricos.

Uma classe recai num conceito expandido de uma estrutura de dados, mas, ao invés de conter apenas dados, pode também abranger funções. Este capítulo demonstra os processos envolvidos na criação de classes, assim como a sua implementação em C++ sob a forma de objetos.

6.1 CONCEITOS GERAIS

O aumento da complexidade das aplicações implicou que o *software* ficasse mais caro e exigiu aos programadores uma maior capacidade de adaptação às novas linguagens e aos sistemas operativos que iam surgindo. As aplicações tornaram-se mais procuradas, é certo, mas também mais dispendiosas e com menor qualidade. Tornou-se urgente encontrar uma forma de aumentar a produtividade, isto é, de fazer com que existisse a possibilidade de se utilizar parte de determinadas aplicações noutras, empregando, basicamente, as mesmas ideias, o mesmo código, os mesmos dados e as mesmas funções.

A tecnologia capaz de lidar com este problema foi a **programação orientada a objetos (POO)**. As características deste tipo de programação foram particularmente bem-sucedidas, na medida em que simplificaram de modo considerável projetos que até então eram demasiado complexos e difíceis de estruturar. Este novo sistema veio conceber técnicas e conceitos de desenvolvimento completamente diferentes. Uma aplicação informática deixou de consistir apenas num conjunto de instruções que visava o desempenho de uma tarefa específica e passou a ser definida como um conjunto de instruções e de objetos que interagem entre si, no qual cada elemento preserva a sua individualidade e desempenha um papel especial na execução de uma tarefa.

A C++ é uma linguagem que permite a automatização de tarefas que envolvem objetos, sendo um objeto uma combinação de código e dados que pode ser manipulada como uma unidade. Exemplos de objetos são ficheiros, diretórios, janelas de linha de comandos, imagens e controlos típicos de interface de utilizador (tais como botões, caixas de seleção e grupos de opções).

Extrapolando para o mundo real, podemos dar um exemplo de como elaborar um modelo de POO tendo como base um elemento principal por nós tão bem conhecido – um retângulo. Este possui duas características fundamentais, base e altura, a partir das

quais é possível, por definição de valores iniciais, calcular a sua área. Assim, num modelo de POO: o retângulo será representado pelo objeto; as características que o definem (base e altura) designam-se por **membros-dados** (ou atributos); e as suas funcionalidades (inicialização e cálculo) designam-se por **membros-função** (ou métodos).

6.2 OBJETOS

As classes são declarações de objetos. Quando definimos um objeto atendendo às suas características e funcionalidades, estamos, na prática, a conceber uma classe. Em C++, as classes são criadas através da palavra-chave `class`, de acordo com a sintaxe seguinte:

```
class classe
{
};
```

Em que `classe` corresponde ao nome que pretendemos atribuir à classe.

Assim, para iniciarmos o processo de conceção do nosso retângulo, basta incluir a sua definição no módulo de código:

```
class Retangulo
{
};
```

6.2.1 ATRIBUTOS

Os atributos são as características que definem os objetos. Exemplos de atributos são o tamanho, a cor e o tipo de letra. Em C++, os atributos de uma classe são definidos no interior desta e a sua declaração é semelhante à das variáveis:

```
class classe
{
    acesso1:
    tipo1 atributo1;
    tipo2 atributo2;
    ...
    acesso2:
    tipo3 atributo3;
    tipo4 atributo4;
    ...
};
```

Em que `acesso` corresponde ao nível de proteção (Tabela 6.1), `tipo`, ao tipo de dados e `atributo`, ao nome de cada atributo.

Este capítulo foca um dos principais conceitos envolvidos na programação orientada a objetos (POO) – a herança. O seu uso permite a reutilização de código, favorecendo a realização de inúmeras tarefas envolvidas no processo de desenvolvimento das aplicações, poupando bastante tempo aos profissionais. Através desta técnica, é possível construir classes baseadas noutras já existentes.

7.1 HERANÇA ÚNICA

Na POO, é possível criar-se uma classe generalista, a partir da qual é permitido desenvolver classes semelhantes. A herança sobrevém quando duas classes têm características e funcionalidades comuns sem serem exatamente iguais.

Extrapolando para o mundo real, podemos dar um exemplo de como elaborar um modelo de POO, tendo como base um elemento principal por nós tão bem conhecido – um veículo. Este possui uma série de características, tais como cor, cilindrada e velocidade a que circula. Além disso, é dotado de um conjunto de funcionalidades, como acelerar, travar e parar. Assim, a partir de uma classe “veículo” é possível produzir diferentes classes que remetam para diversos tipos de veículos em função das suas especificidades, tais como automóveis, motociclos, aviões, helicópteros e navios.

A classe original é considerada de nível superior, sendo designada por **classe-base** (superclasse ou classe-mãe), e a classe gerada a partir da original é considerada de nível inferior, sendo designada por **classe derivada** (subclasse ou classe-filha). Como a classe derivada consegue herdar todos os atributos e métodos da classe-base, a herança apresenta a vantagem de se evitar a repetição desnecessária de código. Por outro lado, é possível definir novos atributos e adicionar outras funcionalidades.

Para derivar uma classe a partir de outra já existente, utiliza-se o sinal de dois pontos (:) para que se estabeleça um grau de parentesco entre a classe-base e a classe derivada:

```
class derivada : acesso base
{
    //Bloco da classe derivada
}
```

Em que derivada é o nome da classe derivada; base, o nome da classe-base; e acesso, as permissões de acesso da classe derivada aos membros da classe-base (Tabela 7.1).

ACESSO	A CLASSE DERIVADA HERDA...
private	Os membros públicos e protegidos como privados.
protected	Os membros públicos e protegidos como protegidos.
public	Exatamente os mesmos níveis de proteção dos membros da classe-base.

TABELA 7.1 – Níveis de proteção de atributos



Caso não seja especificado o nível de proteção, o compilador interpreta-o como sendo `private` (valor por predefinição).

Na Listagem 7.1 são criados dois automóveis (subclasse `Automovel`) e um comboio (subclasse `Comboio`) baseados na superclasse `Veiculo`. Todos os veículos admitem passageiros e suportam uma velocidade máxima; porém, a diferença é que, enquanto um automóvel poderá ser do tipo “ligeiro” ou “pesado”, um comboio é composto por carruagens.

/* Listagem 7.1 – Derivação de classes */

```
#include <iostream>

using namespace std;

//Classe-base
class Veiculo
{
    private:
        int passageiros, velocidade;
    public:
        int get_passageiros()
        {
            return passageiros;
        }
        void set_passageiros(int p)
        {
            passageiros = p;
        }
        int get_velocidade()
        {
            return velocidade;
        }
        void set_velocidade(int v)
        {
            velocidade = v;
        }
};
```

```
//Classe derivada (automóvel)
```

Este capítulo centra-se no estudo de um lote variado de funções matemáticas definidas por uma das bibliotecas-padrão mais comuns da C++ – a **cmath**¹. Para simplificar a compreensão da matéria, as diversas funções serão agrupadas em subcategorias: trigonométricas, exponenciais, de potenciação e de arredondamento.

9.1 FUNÇÕES TRIGONOMÉTRICAS

As funções `sin`, `cos` e `tan` calculam, respetivamente, o seno, o cosseno e a tangente, tal como as respetivas funções inversas `asin`, `acos`, `atan` calculam, respetivamente, o arcosseno, o arcocosseno e a arcotangente dos valores especificados.

A Listagem 9.1 apresenta alguns exemplos de utilização das funções trigonométricas principais (os ângulos são medidos em radianos).

/ Listagem 9.1 – Utilização de funções trigonométricas */*

```
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
    double seno, cosseno, tangente; //Variáveis (funções normais)
    double arcosseno, arcocosseno, arcotangente; //Variáveis (funções inversas)
    //Cálculos
    seno = sin(2); //Seno
    cosseno = cos(3); //Cosseno
    tangente = tan(5); //Tangente
    arcosseno = asin(1); //Arcosseno
    arcocosseno = acos(0.5); //Arcocosseno
    arcotangente = atan(1.5); //Arcotangente
    //Impressão
    cout << seno << endl; //Imprime 0.909297
    cout << cosseno << endl; //Imprime -0.989992
    cout << tangente << endl; //Imprime -3.38052
```

¹ Variante para C++ da biblioteca `math.h`, original da linguagem C.

```
cout << arcosseno << endl; //Imprime 1.5708
cout << arcocosseno << endl; //Imprime 1.0472
cout << arcotangente << endl; //Imprime 0.98274
}
```



Além destas funções trigonométricas principais, estão disponíveis as funções `sinh`, `cosh`, `tanh`, `asinh`, `acosh`, `atanh` para as variantes hiperbólicas das funções senoidal, cossenoidal e tangencial, respetivamente.

9.2 FUNÇÕES EXPONENCIAIS

A função `exp` implementa a função exponencial natural de base e^2 . Já para o cálculo do logaritmo natural (função inversa da exponencial) e do logaritmo de base 10, são utilizadas as funções `log` e `log10`, respetivamente. Estas duas funções aceitam como argumento de entrada um valor igual ou superior a zero.

A Listagem 9.2 coloca em prática a utilização das funções `exp`, `log` e `log10`.

```
/* Listagem 9.2 – Utilização de funções exponenciais */
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
    double e1, e2, log_n, log_10; //Variáveis
    //Cálculos
    e1 = exp(-5); //Exponenciação natural (expoente negativo)
    e2 = exp(5); //Exponenciação natural (expoente positivo)
    log_n = log(3); //Logaritmo natural (expoente positivo)
    log_10 = log10(5); //Logaritmo de base 10 (expoente positivo)
    //Impressão
    cout << e1 << endl; //Imprime 0.00673795
    cout << e2 << endl; //Imprime 148.413
    cout << log_n << endl; //Imprime 1.09861
    cout << log_10 << endl; //Imprime 0.69897
}
```

9.3 FUNÇÕES DE POTENCIAÇÃO

A função `pow` (abreviatura de *power*) permite elevar um número a outro, tendo como argumentos obrigatórios x (a base) e y (o expoente). As funções `sqrt` (abreviatura

² Constante conhecida por número de Euler; número irracional, igual a aproximadamente 2,71828 e usado na representação da função equivalente à função inversa da função exponencial.

Neste capítulo é estudada a biblioteca **cstring**, que possui a definição de variadas funções para classificar e manipular cadeias de caracteres completas ou caracteres individuais.

10.1 MEDIÇÃO

Para se obter o número de caracteres, que são constituintes de uma cadeia de caracteres alfanuméricos, é utilizada a função `length`; para se apagar o seu conteúdo, recorre-se à função `clear`; para se verificar se detém, ou não, algum conteúdo, emprega-se a função `empty`.

A Listagem 10.1 dá um exemplo prático de utilização das funções `length`, `clear` e `empty`.

```
/* Listagem 10.1 – Utilização das funções length, clear e empty */
#include <iostream>
#include <cstring>
using namespace std;
void imprimir(string s); //Protótipo para a função imprimir
int main ()
{
    string cumprimento; //Declaração
    cumprimento = "Olá!"; //Atribuição
    imprimir(cumprimento); //Imprime "Com conteúdo (4 caracteres)"
    cumprimento.clear(); //Limpeza
    imprimir(cumprimento); //Imprime "Sem conteúdo (0 caracteres)"
    cumprimento = "Bom dia!"; //Reatribuição
    imprimir(cumprimento); //Imprime "Com conteúdo (8 caracteres)"
}
void imprimir(string s)
{
    if(s.empty()) //Variável vazia
    {
        cout << "Sem conteúdo";
    }
    else //Variável não vazia
    {
        cout << "Com conteúdo";
    }
}
```

```
    }  
    cout << " (" << s.length() << " caracteres) " << endl;  
}
```

10.2 COMPARAÇÃO

A função `compare`, como o próprio nome indica, compara os conteúdos (totais ou parciais) de duas cadeias de caracteres alfanuméricos. São utilizados os argumentos identificados na Tabela 10.1.

ARGUMENTO	DESCRIÇÃO
<code>pos</code>	Posição inicial no processo de comparação. A indexação é iniciada em 0, pelo que o primeiro caráter é de índice 0, o segundo, de índice 1 e assim sucessivamente.
<code>len</code>	Número de caracteres que deverão ser comparados.
<code>str</code>	Texto de comparação.

TABELA 10.1 – Argumentos da função `compare`



Caso o objetivo seja uma comparação completa entre duas cadeias de caracteres alfanuméricos, apenas é especificado o argumento `str`. Os argumentos `len` e `str` são utilizados apenas em comparações parciais.

A Listagem 10.2 compara dois animais (um gato preto e um gato branco).

/* Listagem 10.2 – Utilização da função `compare` */

```
#include <iostream>  
#include <cstring>  
  
using namespace std;  
  
int main ()  
{  
    string a1 = "gato preto";  
    string a2 = "gato branco";  
    if (a1.compare(a2) != 0)  
    {  
        cout << "Os animais são diferentes\n";  
    }  
    if (a1.compare(0, 4, "gato") == 0)  
    {  
        cout << "O 1.º animal é um gato\n";  
    }  
    if (a2.compare(0, 3, "cão") != 0)  
    {  
        cout << "O 2.º animal não é um cão\n";  
    }  
    if (a1.compare(0, 4, a2, 0, 4) == 0)  
    {  

```

O sistema de ficheiros é, sem dúvida, o aspeto com maior exposição no sistema operativo, estando organizado por uma rede composta por diretórios, subdiretórios e ficheiros, a qual é reconhecida com facilidade por todos os utilizadores. Neste capítulo iremos expor as técnicas atualmente usadas para a gestão do sistema de ficheiros, no sentido de proporcionar ao leitor os conhecimentos envolvidos nos processos de criação, eliminação e renomeação de diretórios e ficheiros através da linguagem C++.

13.1 VERIFICAÇÃO DE ITENS

Para verificar a existência de determinado item do sistema (ficheiro ou diretório), recorre-se à função `stat`, que está definida no cabeçalho `sys/stat.h`. A função `stat` é mencionada em código da seguinte maneira:

```
stat(path, info);
```

Em que `path` corresponde ao caminho (absoluto ou relativo) até ao item que se pretende verificar; e `info`, a um ponteiro para uma estrutura `stat` que permite obter informações específicas acerca do item discriminado no argumento `path`.



A função `stat` retorna 0 ou -1, caso o item exista ou não, respetivamente. Em caminhos absolutos são utilizadas as barras oblíquas (/) para se ir avançando de nível na rede hierárquica de diretórios (por exemplo, `/tmp/dir1`).

A Listagem 13.1 verifica a existência do subdiretório **dir1** no diretório do projeto atual.

/* Listagem 13.1 – Verificação de um diretório */

```
#include <iostream>
#include <sys/stat.h>

using namespace std;

void verificar(const char *caminho)
{
    struct stat info; //Estrutura
    if(stat(caminho, &info) != 0)
    {
        cout << caminho << " não existe";
    }
}
```

```
    else
    {
        cout << caminho << " existe";
    }
}

int main()
{
    verificar("dir1");
}
```



A verificação da existência de ficheiros é feita exatamente do mesmo modo do que as dos diretórios.

13.2 CRIAÇÃO DE DIRETÓRIOS

Para se criar um diretório, recorre-se à função `mkdir`:

```
mkdir(path, mode);
```

Em que `path` corresponde ao caminho do diretório a ser criado; e `mode` às permissões de acesso.



O parâmetro `mode` só é utilizado em ambientes Linux. Consulte os códigos de permissões numéricas em pt.wikipedia.org/wiki/chmod.

A Listagem 13.2 visa a criação do diretório **dir1** no diretório do projeto atual.

```
/* Listagem 13.2 – Criação de um diretório */
#include <iostream>
#include <sys/stat.h>
using namespace std;

int main()
{
    const char *caminho = "dir1"; //Caminho
    struct stat info; //Estrutura
    if(stat(caminho, &info) != 0)
    {
        mkdir(caminho, 777); //Criação do diretório
        cout << caminho << " criado";
    }
    else
    {
        cout << caminho << " já existente";
    }
}
```



Os procedimentos para a criação de ficheiros serão tratados no Capítulo 14.

Em grande parte dos projetos de desenvolvimento, recorre-se a ficheiros de texto para o armazenamento de dados e a realização de tarefas diversas, já que aqueles possuem o denominador comum de tanto os utilizadores como as aplicações poderem ler e entender o seu conteúdo de uma forma bastante rápida e (muitas vezes) conveniente. Neste capítulo iremos focar-nos no acesso e na manipulação de ficheiros de texto com recurso às funções e às classes especificamente projetadas para este tipo de operações.

14.1 CLASSES FUNDAMENTAIS

A linguagem C ++ fornece um conjunto de três classes próprias (Tabela 14.1), definidas na biblioteca **iostream**, que facultam a saída (escrita) e a entrada (leitura) de informação sob a forma de caracteres para e de ficheiros de texto.

CLASSE	LEITURA?	ESCRITA?
<code>ofstream</code>	Sim.	Não.
<code>ifstream</code>	Não.	Sim.
<code>fstream</code>	Sim.	Sim.

TABELA 14.1 – Classes utilizadas no acesso a ficheiros de texto

Como veremos nas próximas secções, o uso destas classes é bastante semelhante ao das classes `cin` e `cout`, mas com a diferença de, ao invés de estipularmos o fluxo de dados de e para os dispositivos-padrão de saída e entrada de informação, utilizarmos ficheiros físicos neste processo.

14.2 ESCRITA

Para escrever em ficheiros de texto, recorre-se à classe `ofstream` (do inglês *output file stream*). Esta classe, além de viabilizar a gravação da informação, é utilizada para abrir (função `open`) e fechar (função `close`) o ficheiro físico.

14.2.1 MÉTODO SUBSTITUTIVO

Na Listagem 14.1 é utilizado um ficheiro de texto para escrever duas linhas. Relativamente à função `open` e à forma como foi invocada: caso o ficheiro não exista, ele será criado; caso exista, todo o conteúdo anterior será substituído. Já a função `is_open` retorna um valor do tipo de dados `bool` (`true`, se o sistema conseguiu abrir o ficheiro, ou `false`, em caso contrário).

/ Listagem 14.1 – Escrita em ficheiros de texto (método substitutivo) */*

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
    ofstream fich; //Declaração
    fich.open ("teste.txt"); //Abertura
    if (fich.is_open())
    {
        fich << "Linha 1" << endl; //Escrita
        fich << "Linha 2" << endl; //Escrita
        cout << "Operação realizada com sucesso";
        fich.close(); //Fecho
    }
    else
    {
        cout << "Operação inválida";
    }
}
```

14.2.2 MÉTODO ADITIVO

Quando o objetivo é proceder à abertura de um ficheiro de texto para acrescentar linhas de dados, deveremos dar explicitamente essa indicação. Para tal, recorre-se à *flag*¹ `app` no segundo argumento da função `open`.



A *flag* `app` encontra-se definida na classe-base `ios` (acrónimo do inglês *input/output stream*).

Na Listagem 14.2 é acrescentada uma linha de texto ao ficheiro **teste.txt** criado na Listagem 14.1.

/ Listagem 14.2 – Escrita em ficheiros de texto (método aditivo) */*

```
#include <iostream>
```

¹ Mecanismo lógico pelo qual uma função (ou outra entidade de programação) pode atuar de modo distinto do predefinido.

Neste capítulo iremos focar os princípios do desenho de interfaces gráficas com base em áreas do conhecimento subjacentes às ciências da computação, nomeadamente a fisiologia, a psicologia, a semiótica e a comunicação (visual, sonora e escrita). Começaremos por abordar os principais conceitos envolvidos na interação homem-computador e terminaremos o capítulo aptos a idealizar o *front-end*¹ de aplicações com características intuitivas, consistentes, objetivas, funcionais e assertivas, proporcionando um encontro real entre as expectativas de usabilidade e a aprovação dos utilizadores finais.

17.1 AS 10 HEURÍSTICAS DE NIELSEN

A norma ISO² 9241 veio estabelecer o conceito de usabilidade como “um conjunto de aspetos a ter em conta para que um produto se adapte a utilizadores específicos, de modo a que estes possam desempenhar tarefas específicas com efetividade, eficiência e satisfação”. A efetividade foca-se nos objetivos primários do processo de interação e repercute-se na maior ou menor facilidade com que os utilizadores executam uma determinada operação, bem como na qualidade dos resultados obtidos.

A eficiência é avaliada pelos erros e desvios que o utilizador comete durante a execução das diferentes tarefas que o *software* proporciona. A satisfação é o espelho do conforto e do agrado dos utilizadores junto das aplicações. No desenvolvimento de *software*, dar resposta a estas três medidas não é uma tarefa simples, uma vez que os profissionais deparam-se com inúmeros aspetos que, à partida, parecem ser secundários, mas que fazem toda a diferença no momento de avaliação global da aplicação em termos de usabilidade. Assim sendo, o pormenor é mais importante do que imaginamos ser.

¹ Parte do *software* que interage diretamente com o utilizador (por exemplo, visualização do conteúdo de uma base de dados e escrita de uma mensagem de *e-mail*); por oposição, o *back-end* refere-se aos processos que os utilizadores não veem, como a introdução efetiva de novos registos numa base de dados e o envio de *e-mails*.

² Do inglês *International Organization for Standardization* (www.iso.org); é uma entidade que estabelece padrões internacionais em variadíssimas áreas, por exemplo, na gestão da qualidade, em medidas e tamanhos e em códigos para representação de países.

Jakob Nielsen³ é considerado o “rei” da usabilidade. Este estatuto deve-se, em grande parte, ao pioneirismo e à sensibilidade visionária que o caracterizam, bem como aos diversos estudos que tem vindo a realizar nesta área. Em 1994, brindou-nos com o seu *best-seller* – *Usability Engineering* –, editado pela Morgan Kaufmann, que constitui uma obra de referência a este nível. Quase todas as metodologias que permitem melhorar a qualidade das interfaces gráficas derivam das 10 heurísticas⁴ que Nielsen descreve no seu livro, e serão estes os aspetos que iremos descrever e analisar ao longo desta secção.

17.1.1 FEEDBACK

A aplicação deve sempre manter os utilizadores informados acerca do que está a acontecer. Do ponto de vista da usabilidade, uma aplicação pode dar dois tipos de *feedback*, dependendo do tipo de tarefa que esteja a desempenhar. Os *feedbacks* de tarefas interruptivas são fornecidos em operações que não permitem o acesso a outras funcionalidades enquanto a atual não for terminada (por exemplo, a gravação de um ficheiro); por sua vez, estamos perante um *feedback* de uma operação não interruptiva em situações nas quais os utilizadores podem executar outras tarefas enquanto a atual estiver a decorrer (por exemplo, o *download* de um ficheiro da Internet). Em termos de apresentação da informação, podemos recorrer a diversos controlos, tais como barras de estado, de notificação e de progresso.

17.1.2 LINGUÍSTICA⁵

Uma aplicação deve “falar” a mesma língua dos utilizadores, ser o mais clara e objetiva possível e privar-se do uso de termos técnicos demasiado complexos e difíceis de compreender. Obviamente, as frases (ou anotações) deverão privilegiar a boa construção sintática, ortográfica e gramatical.

17.1.3 INTERRUPÇÃO

Os utilizadores deverão poder interromper facilmente uma tarefa (e também arrepender-se de a ter levado a cabo). Regra geral, as aplicações incluem os conhecidos

³ Nascido em 1957, em Copenhaga (Dinamarca); é autor, investigador e consultor na área de interfaces de utilização de componentes de *software*.

⁴ Soluções aproximadas para a resolução de problemas; o método heurístico não segue um caminho linear, já que se baseia na intuição, no empirismo e na experimentação.

⁵ Estudo científico da linguagem verbal humana.

Neste capítulo iremos apresentar ao leitor os aspetos essenciais do desenho de interfaces gráficas com recurso à biblioteca GTK¹. Além da criação de janelas básicas, serão estudados os diferentes esquemas gráficos (*layouts*) suportados.

18.1 JANELAS

Em GTK, os objetos que compõem a interface de utilizador, tais como janelas, caixas de diálogo, imagens e controlos, são designados por *widgets* e implementados através da classe `Widget`. O objeto `Window` representa uma janela de aplicação de nível superior, que pode, por sua vez, conter outros *widgets*.

A Tabela 18.1 descreve um conjunto de métodos ao qual o leitor deverá dar especial atenção, dado serem vitais para o desenvolvimento de janelas GTK.

MÉTODO	DESCRIÇÃO
<code>set_title</code>	Define o texto correspondente ao título da janela.
<code>set_size_request</code>	Estabelece o tamanho, em pixéis, da janela.
<code>set_position</code>	Estipula a posição inicial da janela no ecrã. Consulte as constantes utilizadas na secção A.8 do Anexo.
<code>set_resizable</code>	Define se a janela pode (<code>true</code>), ou não (<code>false</code>), ser redimensionada.
<code>set_border_with</code>	Configura a largura das margens, em pixéis, da janela.

TABELA 18.1 – Métodos fundamentais para a criação de janelas GTK



Para que possa testar os exemplos apresentados neste capítulo, deverá instalar e configurar devidamente a biblioteca **gtkmm**, como descrito no Guia de Instalação.

¹ Ou GIMP (acrónimo de *GNU Image Manipulation Program*) Toolkit. Projeto de *software* livre baseado na linguagem C e criado por Peter Mattis, Spencer Kimbal e Josh MacDonald, que inclui um conjunto de ferramentas multiplataforma para o desenvolvimento de interfaces gráficas nos mais variados ambientes. A versão C++ da biblioteca GTK é a `gtkmm`.

É de salientar que neste tipo de aplicações, além de se herdar a janela da classe `Window`, inicializa-se (método `run`) o sistema `GTK` através de uma classe `Main` internamente definida no cabeçalho `gtkmm.h`.

A Listagem 18.1 permite criar uma janela de aplicação com as dimensões de 300 por 200 pixéis, centrada no ecrã, não redimensionável, com margens iguais a 10 pixéis e título personalizado.

/* Listagem 18.1 – Criação de uma janela básica */

```
#include <gtkmm.h>

using namespace Gtk;

class Janela : public Window
{
public:
    Janela(); //Construtor
};

int main(int argc, char *argv[])
{
    Main main(argc, argv); //Aplicação GTK
    Janela janela; //Janela
    main.run(janela); //Inicialização
}

Janela::Janela()
{
    set_size_request(300, 200); //Dimensões
    set_position(WIN_POS_CENTER); //Posicionamento (centro)
    set_resizable(false); //Capacidade de resimensionamento
    set_border_width(10); //Margens
    set_title("Janela GTK"); //Título
}
```

Após a execução do código da Listagem 18.1, deverá visualizar a janela conforme apresentada na Figura 18.1.



FIGURA 18.1 – Primeira aplicação GTK

Neste capítulo pretende-se desenhar uma máquina calculadora de bolso que execute as operações aritméticas básicas de soma, subtração, multiplicação e divisão entre dois números inteiros.

21.1 LAYOUT

Será utilizado um *layout* em grelha (Figura 21.1) que compreenda um *widget* da classe `Entry` (para a apresentação do resultado numa caixa de texto) e os restantes da classe `Button` (para a realização das diferentes operações).



FIGURA 21.1 – Interface para o Projeto I

21.2 DEFINIÇÃO DA CLASSE

Serão utilizadas as seguintes variáveis e métodos:

- `num1`, `num2` – variáveis do tipo `long` que correspondem aos valores atuais do primeiro e do segundo membros envolvidos numa das operações aritméticas;
- `opera` – variável do tipo `string` que contém o símbolo da operação escolhida pelo utilizador;

- ⊙ `res` – variável auxiliar booleana que indica se o valor atualmente mostrado no ecrã corresponde ao resultado de uma operação (valor `true`) ou a um dos membros da operação (valor `false`);
- ⊙ `void digitos_clicked(int digito)` – método associado aos eventos de clique dos botões de dígitos;
- ⊙ `void operacoes_clicked(int operacao)` – método associado aos eventos de clique dos botões de operações;
- ⊙ `void calc()` – método associado ao evento de clique do botão **CALCULAR**;
- ⊙ `void limpar()` – método associado ao evento de clique do botão **CLR**.



Para evitar a repetição de código, são utilizados *arrays* de objetos para os botões de dígitos e de operações. Por outro lado, e como a aplicação prevê que os membros das operações serão superiores ou iguais a 0, as variáveis `num1` e `num2` são inicializadas com -1.

Assim, comece por definir a sua classe em concordância com a Listagem 21.1.

/* Listagem 21.1 – Definição da classe */

```
#include <gtkmm.h>
#include <string>

using namespace Gtk;
using namespace std;

class Janela : public Window
{
public:
    Janela(); //Construtor
private:
    //Container
    Grid grelha;
    //Variáveis (inicializações)
    long num1 = -1;
    long num2 = -1;
    string opera = "";
    bool res = false;
    //Objetos
    Entry mostrador;
    Button digitos[10];
    Button operacoes[4];
    Button clr;
    Button calcular;
    //Eventos
    void digitos_clicked(int digito);
    void operacoes_clicked(int operacao);
    void calc();
```

Neste capítulo iremos desenvolver uma modalidade de um jogo muito conhecido das salas de casino – o *Video Poker*.

22.1 LAYOUT

As primeiras máquinas de *Video Poker* foram construídas na década de 70 do século XX, mas só obtiveram popularidade na década seguinte, altura em que foram amplamente difundidas pelos luxuosos casinos de Las Vegas, nos EUA.

Há várias modalidades de *Video Poker*, mas optaremos pela mais conhecida – o *Jacks or Better* –, em que os jogadores só têm direito a prémio se conseguirem, pelo menos, um par de valetes, rainhas, reis ou ases (por exemplo, dois ternos ou duas manilhas não dão prémio).

Iremos recorrer a um *layout* de posicionamento fixo. A Figura 22.1 ilustra o resultado de toda a programação do jogo.



FIGURA 22.1 – Interface para o Projeto II

22.2 REGRAS

O jogador começa com 100 moedas (créditos) e por cada tentativa é-lhe descontada uma moeda. Para começar a jogar, faz-se clique no botão **DEAL** e surgem cinco cartas diferentes de um baralho único; depois, o jogador só tem de aplicar a sua estratégia,

selecionando as cartas que pretende manter (podendo preservar e/ou substituir todas ou apenas parte delas). O prémio alcançado (Tabela 22.1) no final de cada “volta” depende da combinação que o jogador obteve após o pedido de substituição de cartas (botão DRAW).


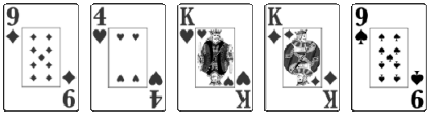
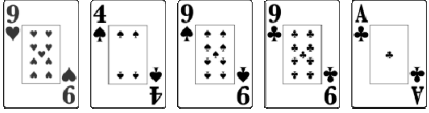
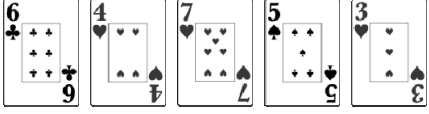
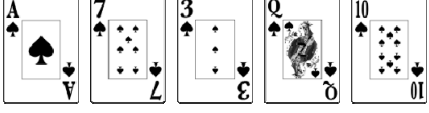
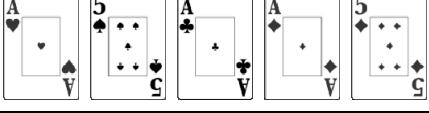
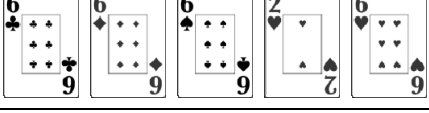
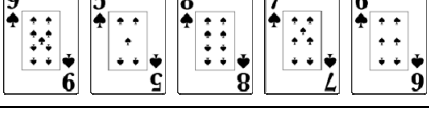
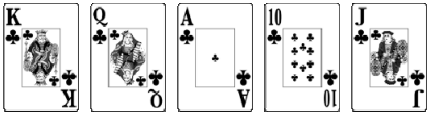
PRÊMIO	DESCRIÇÃO	EXEMPLO
<i>Jacks or Better</i> (+1 crédito)	Par de valetes, rainhas, reis ou ases	
<i>Two Pair</i> (+ 2 créditos)	Quaisquer dois pares	
<i>Three of a Kind</i> (+ 3 créditos)	Três cartas do mesmo valor (tripla)	
<i>Straight</i> (+ 4 créditos)	Sequência (de naipes diferentes)	
<i>Flush</i> (+ 6 créditos)	Todas as cartas do mesmo naipe	
<i>Full House</i> (+ 9 créditos)	Um par + uma tripla	
<i>Four of a Kind</i> (+ 25 créditos)	Quatro cartas com o mesmo valor	
<i>Straight Flush</i> (+ 50 créditos)	Sequência (do mesmo naipe)	
<i>Royal Flush</i> (+ 250 créditos)	Maior sequência possível do mesmo naipe (com ás, rei, rainha, valete e 10)	

TABELA 22.1 – Tabela de prémios do jogo *Jacks or Better*