

C++ Multiplataforma e Orientação a Objetos

versão preliminar

Essa versão não foi 100% revista, mas está sendo liberada a pedido de muitos alunos.

Por Sergio Barbosa Villas-Boas (sbVB)

Email: sbvb@sbvb.com.br

URL: www.sbvb.com.br

Versão 7.0, de 23 de Março de 2006

Consultor em tecnologia e estratégia para desenvolvimento de software multiplataforma, baseado em tecnologias gratuitas e padrões abertos.



<http://www.sbvb.com.br>

Sobre o Autor

Sergio Barbosa Villas-Boas atua como Professor Adjunto no DEL (Departamento de Engenharia Eletrônica e Computação da UFRJ), desde 1987. Sua carreira sempre esteve ligada a desenvolvimento de software. Já trabalhou com instrumentação e controle, em sistemas em tempo real, em simulação numérica, sistemas cliente-servidor, sistemas GUI multiplataforma (com wxWidgets), segurança de informação, sistemas web, integração de sistemas e XML (principalmente com Xerces), sistemas corporativos, sistemas baseados em componentes, objetos distribuídos, uso de software gratuito, entre outros assuntos. Ultimamente tem havido ênfase no desenvolvimento de sistemas usando o paradigma orientado a objetos.

Desde a volta do doutorado no Japão, em 1998, tem se dedicado ao ensino na universidade, e ao continuado aprendizado por participar de diversos cursos e conferências. Além disso, dedica-se a projetos de extensão universitária, em geral com consultoria especializada em parcerias com diversas empresas.

Participou e participa diretamente de diversos projetos de desenvolvimento de software, especialmente para COTS (Commercial Off-The-Shelf, ou “produto de prateleira”), usando principalmente C++ multiplataforma e também java.

Desde 2000, mantém o site www.sbv.com.br, onde divulga o material de diversos cursos que ministra.

Esse livro está sendo re-editado em função de vários pedidos, mas não é o material mais recente e revisado do autor a respeito de C++. Os slides do curso disponíveis no site estão mais atualizados que esse livro.

Índice

Sobre o Autor.....	2
Índice	3
1 Introdução	14
1.1 Prefácio	14
1.2 Histórico do livro.....	19
1.3 Como usar o livro	21
2 Tecnologia da informação e negócios.....	24
2.1 Experiência de vida e visão de negócios.....	24
2.2 Negócios forjam a tecnologia.....	27
2.3 Classificação dos atores do ambiente para TI	29
2.4 Ambiente econômico e modelo de negócio.....	30
2.5 Web impacta ambiente econômico	31
2.6 Analogia entre militarismo e negócios.....	32
2.6.1 Inteligência 33	
2.7 Conhecimento empacotado	34
2.8 O produto “software”	36
2.9 Analogia entre biologia e negócios	38
2.9.1 Zona de conforto 39	
2.10 Resultado da competição: resta um ou poucos.....	40
2.11 Melhores práticas.....	40
2.12 Incorporação de novidades.....	41
2.13 Mercadoria e produto diferenciado	44
2.13.1 Mercadoria escassa 45	
2.13.2 Estratégia lucrativa 46	
2.14 O objetivo é atender ao cliente ?.....	47
2.15 Vantagem competitiva.....	49
2.16 Sociedade do conhecimento	50
2.16.1 Mudança do paradigma de negócios 52	
2.16.2 Mercadorização 53	
2.16.3 Remuneração × agregação de valor na cadeia produtiva 54	
2.17 Políticas de apoio a “economia do software”	54

2.17.1	O que o governo pode fazer	54
2.17.2	O que os indivíduos podem fazer	56
2.17.3	O que as universidades podem fazer	56
2.17.4	O que as empresas podem fazer	57
2.18	Livro no Século 21	58
2.18.1	O modelo tradicional	58
2.18.2	Mudanças no ambiente	59
2.18.3	O novo modelo	61
2.18.4	Leitor gosta de ter livro em formato eletrônico	63
2.18.5	Aferição de mérito acadêmico	65
2.18.6	Quem está usando o modelo novo	66
2.18.7	Idioma Português no mundo	66
2.19	Tribos de tecnologia	67
2.20	A escolha tecnológica é uma escolha estratégica	68
2.21	Java × C++	70
2.21.1	Porque sim Java	70
2.21.2	Porque não Java	71
2.21.3	Porque sim C++	73
2.21.4	Porque não C++	74
2.22	Estratégia para integradores de solução	75
2.22.1	Análise SWOT	78
2.23	Resumo	80
2.24	Estratégia e C++	83
2.25	Humor	84
2.26	Exercícios	84
3	Conceitos introdutórios	87
3.1	Breve história do C/C++	87
3.2	Classificação de interfaces de programas	89
3.3	Programando para console	90
3.4	Linguagens de programação de computador	90
4	Conheça o Seu Compilador	92
4.1	Visual C++ 6.0 SP5	92
4.1.1	Reconhecendo o Compilador	92
4.1.2	“Hello world” para DOS	98
4.1.2.1	Adicionando argumentos para a linha de comando	101

4.1.3	Usando o Help	102
4.1.4	Projetos (programas com múltiplos fontes)	103
4.1.5	Bibliotecas	104
4.1.5.1	Fazer uma biblioteca.....	104
4.1.5.2	Incluir uma biblioteca num projeto	106
4.1.5.3	Examinar uma biblioteca	106
4.1.6	Debug	106
4.1.7	Dicas extras	108
4.1.7.1	Macro para formatação do texto fonte.....	108
4.1.7.2	Acrescentando Lib no Project Settings	125
4.1.7.3	Class View.....	126
4.1.7.4	Usando bibliotecas de ligação dinâmica (DLL)	127
4.1.7.5	DLL para Windows.....	131
4.1.7.6	Otimização do linker para alinhamento de código	131
4.1.8	Detectando vazamento de memória	131
4.2	Borland C++ builder 5.0.....	133
4.2.1	Reconhecendo o Compilador	133
4.2.2	“Hello world” para DOS	134
4.2.2.1	Adicionando argumentos para a linha de comando	136
4.3	C++ para win32 gratuito	137
4.3.1	MinGW	137
4.3.2	djgpp	137
4.4	Dev-C++ 4.0	138
4.4.1	Reconhecendo o Compilador	138
4.4.2	“Hello world”	139
4.4.2.1	Adicionando argumentos para a linha de comando	140
4.4.3	Usando o Help	142
4.4.4	Projetos (programas com múltiplos fontes)	142
4.5	g++ (do unix)	142
4.5.1	“Hello world”	143
4.5.1.1	Adicionando argumentos para a linha de comando	143
4.5.2	Usando o Help	143
4.5.3	Projetos (programas com múltiplos fontes)	143
4.5.4	Bibliotecas	144
4.5.4.1	Incluir uma biblioteca num projeto	144

4.5.4.2	Fazer uma biblioteca.....	144
4.5.4.3	Examinar uma biblioteca	145
4.5.5	Fazendo uma biblioteca usando libtool	145
4.5.5.1	Instalando uma biblioteca dinâmica	146
4.5.6	Debug	148
4.5.7	Definindo um identificador para compilação condicional	148
4.5.8	O pacote RPM do linux	149
4.5.8.1	rpm binário e rpm com fonte.....	149
4.5.8.2	Alguns comandos do rpm	149
4.5.8.3	Construindo um rpm.....	150
4.5.8.3.1	Introdução	150
4.5.8.3.2	O header	151
4.5.8.3.3	Preparação (prep)	152
5	Princípios de C/C++	154
5.1	O primeiro programa.....	154
5.2	Formato livre	154
5.3	Chamada de função	155
5.4	Declaração e definição de funções.....	155
5.5	Comentários.....	156
5.6	Identificador.....	156
5.7	Constantes literais	157
5.8	Escopo	157
5.9	Tipos de dados padrão (<i>Standard Data Types</i>)	158
5.10	Palavras reservadas do C++ (keywords)	158
5.11	Letras usadas em pontuação.....	159
5.12	Letras usadas em operadores.....	159
5.13	Exercícios resolvidos	159
5.13.1	Resposta	160
6	Estrutura do Compilador.....	161
6.1	Entendendo o Compilador.....	161
6.2	Protótipos (<i>prototypes</i>)	162
6.3	Projetos em C/C++	164
6.4	Header Files (*.h)	166
6.5	Biblioteca (<i>library</i>)	167
6.5.1	Utilizando Bibliotecas prontas	167

6.5.2	Fazendo bibliotecas	168
6.6	Regras do compilador	169
7	Linguagem C/C++	171
7.1	Chamada de função por referência e por valor	171
7.2	Tipos de dados definidos pelo programador	172
7.3	Maquiagem de tipos (<i>type casting</i>)	173
7.4	Operações matemáticas	174
7.5	Controle de fluxo do programa	174
7.6	Execução condicional.....	175
7.7	Laços (<i>loop</i>) de programação.....	175
7.7.1	Laço tipo “do-while”	175
7.7.2	while	176
7.7.3	for	176
7.7.4	Alterando o controle dos laços com break e continue	177
7.7.5	Exercício	178
7.8	switch-case	178
7.9	arrays	179
7.10	Ponteiros.....	181
7.10.1	Ponteiro para ponteiro	182
7.11	Arrays e ponteiros.....	184
7.12	Arrays multidimensionais	184
7.13	Parâmetros da função main.....	184
7.14	Compilação condicional.....	186
7.15	Pré processador e tokens (símbolos) usados pelo pré-processador.....	186
7.16	#define	186
7.17	operador #	187
7.18	operador ##	187
7.19	Número variável de parâmetros.....	187
7.20	Exercícios	188
8	Técnicas para melhoria de rendimento em programação.....	190
8.1	Reutilização de código	190
8.2	Desenvolvimento e utilização de componentes.....	192
8.3	Programação estruturada	193
9	Programação orientada a objeto	195
9.1	Conceitos básicos.....	195

9.2	Nomenclatura para paradigma procedural e para paradigma OO	195
9.3	Representação gráfica de classes	196
9.4	Objetos.....	197
9.4.1	Uso de classes (funções dentro de estruturas)	198
9.4.2	Operador de escopo	199
9.5	Polimorfismo	200
9.5.1	Argumento implícito (<i>default argument</i>)	200
9.6	Análise, projeto e programação OO	201
9.7	Exemplo conceitual sobre herança	201
9.8	Herança.....	203
9.9	Herança múltipla e classe base virtual	210
9.10	União adiantada × união tardia (<i>early bind</i> × <i>late bind</i>)	213
9.10.1	Classe abstrata	215
9.10.2	Porque usar união tardia ?	219
9.11	Construtor e destrutor de um objeto.....	220
9.11.1	Default constructor (construtor implícito)	222
9.11.2	Ordem de chamada de construtor e destrutor para classes derivadas	223
9.11.3	Inicialização de atributos com construtor não implícito	225
9.11.4	Construtor de cópia (<i>copy constructor</i>) e operator=	228
9.11.5	Destrutores virtuais	229
9.11.6	Array de objetos	231
9.11.7	Array de objetos e construtor com parâmetros	232
9.12	Polimorfismo - sobrecarga de operadores.....	233
9.12.1	Operador de atribuição (assignment operator – operator=)	233
9.12.2	Porque redefinir o construtor de cópia e operador de atribuição ?	233
9.12.3	O único operador ternário	234
9.12.4	Operadores binários e unários	235
9.13	this	239
9.14	lvalue	239
9.15	Encapsulamento de atributos e métodos	240
9.15.1	friend	244
9.16	Alocação de Memória.....	246
9.16.1	Vazamento de memória	247
9.16.2	Objetos com memória alocada	248
9.16.3	Array de objetos com construtor não padrão	251

9.17	Criando uma nova classe	252
10	Biblioteca padrão de C++	255
10.1	Introdução.....	255
10.2	Entrada e saída de dados pelo console.....	255
10.3	Sobrecarga do operador insersor (<<) e extrator (>>)	256
10.4	Formatando Entrada / Saida com streams.....	258
10.4.1	Usando flags de formatação	259
10.4.2	Examinando os flags de um objetos	260
10.4.3	Definindo o número de algarismos significativos	261
10.4.4	Preenchendo os espaços vazios com o método fill	262
10.4.5	Manipuladores padrão	262
10.4.6	Manipuladores do usuário	263
10.4.7	Saída com stream em buffer	264
10.5	Acesso a disco (Arquivos de texto para leitura/escrita).....	265
10.5.1	Escrevendo um arquivo de texto usando a biblioteca padrão de C++	266
10.5.2	Escrevendo um arquivo de texto usando a biblioteca padrão de C	266
10.5.3	Lendo um arquivo de texto usando a biblioteca padrão de C++	267
10.5.4	Dica para leitura de arquivo de texto.	268
10.5.5	Lendo um arquivo de texto usando a biblioteca padrão de C	268
10.6	Acesso a disco (Arquivos binários para leitura/escrita)	269
10.6.1	Escrevendo um arquivo binário usando a biblioteca padrão de C++	269
10.6.2	Escrevendo um arquivo binário usando a biblioteca padrão de C.	269
10.6.3	Lendo um arquivo binário usando a biblioteca padrão de C++	270
10.6.4	Lendo um arquivo binário usando a biblioteca padrão de C	270
11	Tratamento de exceção (<i>exception handling</i>)	272
12	RTTI – Identificação em tempo de execução	276
12.1	Introdução.....	276
12.2	Configurando o compilador para RTTI.....	276
12.3	Palavras reservadas para RTTI.....	276
12.4	typeid.....	276
12.5	typeid e late bind	277
13	namespace	278
13.1	Introdução.....	278
13.2	Namespace aberto	280
13.3	Biblioteca padrão de C++ usando namespace.....	281

13.4	Adaptando uma biblioteca existente para uso de namespace std	282
14	Programação para web com C++ e VBMcgi.....	286
14.1	Introdução.....	286
14.2	Navegando na teia	288
14.2.1	CGI	290
14.3	Escolha de uma tecnologia para web/cgi	291
14.4	História e futuro da VBMcgi	293
14.5	Instalando VBMcgi	294
14.5.1	Windows, Visual C++	294
14.5.2	Windows, Borland Builder C++ 5.0	298
14.5.3	Unix, Gnu C++ compiler	300
14.6	Usando VBMcgi.....	301
14.6.1	Programas CGI e web server	301
14.6.2	Programas CGI e o ambiente (environment)	305
14.6.3	Hello VBMcgi	309
14.6.4	Decodificando o formulário	309
14.6.4.1	textbox (caixa de texto).....	310
14.6.4.2	textarea (área de texto).	311
14.6.4.3	checkbox (caixa de checagem)	312
14.6.4.4	radio button (botão radial)	313
14.6.4.5	drop-down (deixar escapular para baixo)	315
14.6.4.6	Exemplo completo na web.....	316
14.6.5	Usando a funcionalidade “string change”	316
14.6.5.1	Adicionando implicitamente strings.....	319
14.6.6	Usando a funcionalidade “call function”	320
14.6.6.1	Passando parâmetros para a função da “call function”	323
14.6.7	Redirecionando um programa cgi	326
14.6.8	Cookies	327
14.6.8.1	Configuração do navegador para ver os cookies	328
14.6.8.2	Expiração	328
14.6.8.3	Domínio de um cookie	330
14.6.8.4	Sistema na web com login usando cookies.....	330
14.6.9	Programas cgi com dados não html	334
14.7	Contador de página	334
14.8	Calendário	336

14.9	Menu em arquivo	338
14.10	Programação em 3 camadas e sistemas na web com VBMcgi.....	342
15	Programação genérica (template)	344
15.1	Analogia entre template e fôrma de bolo	344
15.2	Programação genérica por exemplos	345
15.2.1	Função genérica	345
15.2.2	Classe genérica	346
15.2.3	Algoritmos genéricos	348
15.3	Classes genéricas contenedoras.....	352
15.3.1	VBList: uma lista encadeada genérica	352
15.4	VBMath - uma biblioteca de matemática matricial em genérico.....	358
16	STL - Standard Template Library	364
16.1	Introdução.....	364
16.1.1	Classes contenedoras	365
16.1.2	Classes iteradoras	365
16.1.3	Algoritmos	365
16.2	Preâmbulo	365
16.2.1	Classes e funções auxiliares	367
16.2.1.1	Par (pair)	367
16.2.2	Operadores de comparação	368
16.2.3	Classes de comparação	369
16.2.4	Classes aritméticas e lógicas	371
16.2.5	Complexidade de um algoritmo	371
16.2.5.1	Algumas regras.....	372
16.3	Iterador (iterator)	373
16.3.1	Iteradores padrão para inserção em contenedores	373
16.4	Contenedor (container)	375
16.4.1	Vector	375
16.4.2	List	376
16.4.3	Pilha (Stack)	377
16.4.4	Fila (Queue)	377
16.4.5	Fila com prioridade (priority queue)	379
16.4.6	Contenedores associativos ordenados	380
16.4.6.1	Set.....	381
16.4.6.2	Multiset.....	383

16.4.6.3	Map.....	383
16.4.6.4	Multimap.....	385
16.5	Algoritmos.....	385
16.5.1	remove_if.....	385
17	Componentes de Programação.....	388
17.1	Para Windows & DOS.....	388
17.1.1	Programa para listar o diretório corrente (Visual C++)	388
17.1.2	Porta Serial	389
17.1.3	Porta Paralela	392
17.2	Componentes para unix (inclui Linux).....	394
17.2.1	Programa para listar o diretório corrente	394
17.2.2	Entrada cega (útil para entrada de password)	398
17.3	Elementos de programação em tempo real.....	399
17.3.1	Conceitos de programação em tempo real	399
17.3.2	Programa em tempo real com “status loop”	399
17.3.3	Programa em tempo real com interrupção	403
17.3.4	Programas tipo “watch dog” (cão de guarda)	405
18	Boa programação × má programação.....	406
18.1	Introdução.....	406
18.2	Itens de boa programação.....	407
18.2.1	Indentação correta	407
18.2.2	Não tratar strings diretamente, mas por classe de string	407
18.2.2.1	Motivos pelos quais é má programação tratar strings diretamente	408
18.2.2.2	Solução recomendada para tratamento de strings: uso de classe de string	409
18.2.3	Acrescentar comentários elucidativos	411
18.2.4	Evitar uso de constantes relacionadas	411
18.2.5	Modularidade do programa	412
18.2.6	Uso de nomes elucidativos para identificadores	413
18.2.7	Programar em inglês	413
19	Erros de programação, dicas e truques.....	416
19.1	Cuidado com o operador , (vírgula)	416
19.2	Acessando atributos privados de outro objeto	416
19.3	Entendendo o NaN	417
19.4	Uso de const_cast	418

19.5	Passagem por valor de objetos com alocação de memória.....	419
19.6	Sobrecarga de insersor e extrator quando se usa namespace	420
19.7	Inicializando membro estático de uma classe	421
19.8	Alocação de array de objetos não permite parâmetro no construtor	422
19.9	Ponteiro para função global e ponteiro para função membro (ponteiro para método)	422
19.10	SingleTon.....	423
19.11	Slicing em C++	424
19.12	Uso desnecessário de construtores e destrutores	426
19.13	Dicas de uso de parâmetro implícito.....	427
20	Incompatibilidades entre compiladores C++	429
20.1	Visual C++ 6.0 SP5	429
20.1.1	for não isola escopo	429
20.1.2	Comportamento do ifstream quando o arquivo não existe	429
20.1.3	ios::nocreate não existe quando fstream é usado com namespace std	430
20.1.4	Compilador proíbe inicialização de variáveis membro estáticas diretamente	430
20.1.5	“Namespace lookup” não funciona em funções com argumento	430
20.1.6	Encadeamento de “using” não funciona	431
20.1.7	Erro em instanciamento explícito de funções com template	431
20.2	Borland C++ Builder (versão 5.0, build 12.34).....	431
20.2.1	Inabilidade de distinguir namespace global de local	431
20.2.2	Erro na dedução de funções com template a partir de argumentos const	431
20.2.3	Erro na conversão de “const char *” para “std::string”	432
20.3	Gnu C++ (versão 2.91.66)	432
20.3.1	Valor do elemento em vector<double> de STL	432
21	Bibliografia.....	433
21.1	Livros.....	433
21.2	Páginas web	438

1 Introdução

1.1 Prefácio

A linguagem de programação C/C++ é um assunto já relativamente tradicional no dinâmico mercado de tecnologia de informação, sempre cheio de novidades. Existe um acervo bibliográfico substancial sobre esse assunto. Portanto surge a pergunta: ainda há espaço ou necessidade para um novo livro sobre C/C++ ? Acredito que sim, pelos motivos abaixo.

1. Deve-se discutir a opção de uso de C/C++ como uma escolha estratégica.

Mesmo que não seja gestor de uma empresa, o indivíduo precisa gerenciar a própria carreira. Mesmo que não se trabalhe com vendas, é preciso ser capaz de vender a própria força de trabalho no mercado. É muito conveniente para o profissional de tecnologia de informação desenvolver minimamente o conhecimento sobre estratégia (empresarial e pessoal). Para desenvolver esse tipo de pensamento e atitude, deve-se discutir elementos que auxiliem a interpretação o mundo a partir de uma visão de negócios, particularmente negócios relacionados a tecnologia de informação. Inúmeros negócios e inovações na área de tecnologia envolvem o uso de C/C++, mas curiosamente nunca vi um livro tutorial sobre C/C++ que discutisse minimamente sobre o ambiente de negócios, e o relacionando da escolha de tecnologia e de estratégia pessoal e empresarial. A escolha da linguagem C/C++ deve (ou não) ser feita, com conhecimento das vantagens e desvantagens dessa escolha.

Na primeira parte deste livro, a partir da página 23, discute-se o relacionamento entre tecnologia e negócios, escolha tecnológica, estratégia de indivíduos empresas, e assuntos relacionados. Como consequência dessa reflexão, analisa-se a escolha da linguagem C/C++ ou alternativas, verificando vantagens e desvantagens.

2. Deve-se mostrar que C/C++ é opção concreta para programação para web/cgi.

Ainda há MUITO o que fazer no mundo a partir do desenvolvimento de sistemas com tecnologia de web/cgi. Quando a web conquistou para o mundo, a partir de cerca de 1995, eu já trabalhava há muito tempo com C/C++. Tive a expectativa que a web seria mais uma novidade no setor de tecnologia de informação, para a qual se poderia desenvolver software com a linguagem C/C++. No início, fiquei incomodado pela dificuldade de se encontrar referências boas sobre como programar para web/cgi com C/C++.

A expectativa de ser possível usar C/C++ para web/cgi é correta, apenas o material tutorial e as bibliotecas de apoio ainda não eram boas o suficiente. Resolvi então estudar e pesquisar sobre assunto, fazendo o trabalho convergir para uma biblioteca e um material tutorial para desenvolvimento de sistemas web/cgi baseado em C++. O resultado é a biblioteca gratuita VBMcgi [45], que permite desenvolvimento de software para web/cgi usando C++ multiplataforma. Esse desenvolvimento é feito com uma arquitetura de software elegante, em que se garante isolamento entre a camada de apresentação (onde o webdesigner usar html) e a camada de implementação das regras de negócio (onde o webmaster usa C++ e VBMcgi).

Quem usa ou quer usar C++ não precisa aprender outra linguagem para desenvolver software para web/cgi. O uso da biblioteca VBMcgi permite o desenvolvimento de software para web em C++ com arquitetura em 3 camadas. Isso leva o que leva ao desenvolvimento de sistemas com boa *manutibilidade* (isto é, pode-se fazer manutenção facilmente).

3. **C/C++ deve ser ensinado como multiplataforma.** Ainda há relativamente poucos livros que enfatizam o aspecto multiplataforma dessa linguagem. Muitos livros se propõe a ensinar C/C++ a partir de um compilador específico. Considero isso em geral um erro. A ênfase no ensino de C/C++ deve ser em cima dos conceitos, e não no compilador ou no sistema operacional. No capítulo “Conheça o Seu Compilador” (página 23), há um conjunto de explicações sobre operacionalidade dos principais compiladores para Windows e Unix. Além disso, há um capítulo sobre “Diferenças entre compiladores” (página 426), onde são listadas pequenas (porém importantes) diferenças de comportamento entre os compiladores.
4. **C/C++ deve ser ensinado com ênfase em Orientação a Objetos.** Os conceitos de desenvolvimento de sistemas orientados a objetos não estão maduros ainda. Apesar de todas as publicações sobre desenvolvimento de sistemas usando paradigma orientado a objetos, nem todas as questões foram resolvidas ainda. Por exemplo: banco de dados orientados a objetos é algo ainda precisa provar que tem vantagens em relação a banco de dados relacionais (desses que usam SQL). Mas mesmo assim, é basicamente claro que a linguagem C++ e o paradigma OO são em muitos casos método superior de desenvolvimento se comparado à linguagem C e paradigma procedural. Embora C++ tenha surgido depois de C, não é em geral uma boa abordagem didática ensinar a um jovem profissional C “puro” antes de C++. Isso porque seria uma grande perda de tempo (para dizer o mínimo). A melhor abordagem didática é ensinar C++ diretamente, com ênfase em orientação a objetos, e apenas mencionar C quando for preciso.

O fato de haver um legado extenso sobre C (literatura, exemplos, etc.), e de que C++ tem compatibilidade reversa com C (isso é, em geral um programa C é também um programa C++), faz com que muita literatura tradicional seja na prática algo contra-indicado para um profissional jovem. A ênfase em C++ e OO, em detrimento de C, deve ser a ênfase de um livro didático moderno. Ainda há relativamente poucos livros com essa ênfase.

5. **Deve-se pensar em produzir um livro num modelo inovador, compatível com o século 21.** Apesar de toda a modernidade, e de toda a impressionante produção de obras de literatura técnica, o modelo de negócios na produção de livros curiosamente pouco evoluiu em relação ao modelo tradicional. Esse livro está sendo produzido e distribuído num modelo inovador, compatível com o ambiente do século 21. Leia mais sobre assunto na seção 2.18.
6. **Deve-se ensinar regras práticas para boa programação.** Ainda há relativamente poucos livros de C/C++, principalmente em Português, que enfatizam a boa programação, reutilização de código e o trabalho em equipe. Uma abordagem simplória da atividade do profissional de TI é acreditar que “programa de boa qualidade é aquele que funciona”. Essa abordagem está superada hoje. Há valorização do processo de desenvolvimento, na arquitetura e na manutibilidade do software. Há interesse no ciclo de vida do produto que se está desenvolvendo. De uma forma geral, entende-se que quando o processo de desenvolvimento é feito dentro de certas características, a qualidade do produto final é melhor. Isso se reflete na programação, onde se espera que o profissional siga “regras de boa programação”, e que tenha atitude de usar (e eventualmente desenvolver) componentes de software.

No capítulo “Boa programação × má programação” (página 406), enfatiza-se os aspectos que um programa deve ter para atingir as necessidades práticas de empresas e organizações. Nesse capítulo, o objetivo de se obter reutilização de código, trabalho em equipe, bem como métodos práticos para minimizar a possibilidade de bugs é traduzido em regras escritas.

Além disso, o conhecimento consolidado de como se desenvolvem e se usam bibliotecas e componentes, discutido na parte 2 do livro, permite ao programador usar com destreza bibliotecas e componentes feitas por outras pessoas, e com isso melhorar muito seu desempenho de programação. Eventualmente pode-se também trabalhar no sentido inverso, isto é, desenvolvendo bibliotecas para que outros usem, e considerar a biblioteca como o produto da empresa.

7. **Deve-se produzir obras originais em Português.** É amplamente aceito o fato de que atualmente o centro do desenvolvimento de tecnologia de

informação produz obras originais no idioma Inglês. Muitos livros são produzidos originalmente em Inglês, e posteriormente são traduzidos para Português e outros idiomas. Qualquer indivíduo ou empresa que pretenda ter seu trabalho de computação reconhecido no atual momento do mundo, com pensamento global, deve ter conteúdo produzido em Inglês. Portanto, produzir uma obra original em Português seria investir num idioma fora do centro de progresso, e portanto um “desperdício de energia” (pois se poderia estar investindo para produzi-lo em Inglês, com visibilidade para todo o mundo). Enquanto me dedico a esse livro, observo alguns colegas professores de universidade que se empenham por produzir livros em Inglês, o que merece muito respeito.

O fato de produzir uma obra original como essa em Português é portanto, antes de tudo, um ato de amor ao nosso país e a nossa cultura. É também um ato de fé nas chances do Brasil (e nos países que falam Português) de assumir um papel mais sólido no mercado mundial de software e assuntos relacionados. Nosso povo tem reconhecidas características de criatividade e flexibilidade – muito valorizadas na atividade de desenvolvimento de software.

Na seção 2.18.7, na página 66, fala-se mais sobre a situação do idioma Português no mundo. Na seção 18.2.7, na página 413, discute-se sobre a escolha do idioma para a programação em si. Apesar de esse livro ser produzido originalmente em Português, quase sempre se escreve código fonte no idioma Inglês. Isso não é uma contradição, mas uma forma de ação recomendada a partir de uma análise ponderada do ambiente, e de formulação de estratégia.

- a. **Idioma inglês nos códigos fonte.** Considerando-se combinadamente alguns fatores já abordados, conclui-se que o melhor idioma para os programas fonte de exemplo é inglês. O desenvolvedor deve trabalhar com atenção ao mesmo tempo em tecnologia e em negócios. No mundo globalizado, empresas são compradas por outras empresas a todo momento. Seja uma empresa A, cujo produto é um software. Seja uma empresa maior B, que se interessa em comprar a empresa A (menor que B, mas apresentando bom crescimento). O processo de a empresa compradora B avaliar no valor da empresa A inclui a observação do código fonte. Se o código fonte é em Inglês, a empresa A passa a ser vendável mundialmente. Se o desenvolvedor e o empreendedor entendem que escrever o código fonte de seus programas em inglês é o que deve ser feito, então o material tutorial deve ensinar e induzir o profissional a trabalhar dessa forma. Mas como se deseja produzir um material em Português, então o livro deve ter explicações em texto corrido em Português, mas ter os códigos fonte de exemplo em Inglês.

Convém comentar que como autor, sou também consumidor de livros sobre desenvolvimento de software. Qualquer livro sobre tecnologia atualmente é produzido em relativamente pouco tempo, e isso faz aumentar a probabilidade de erro. Um livro sobre software traduzido de Inglês para Português, cheio de códigos fonte, em que o tradutor opte por traduzir esse código fonte, faz aumentar muito a probabilidade de haver erro no livro. Isso porque é quase certo que o tradutor jamais compilou o texto que apresenta no livro (mas o autor original em geral faz isso). Em outras palavras: além de poder ser considerado conceitualmente inadequado desenvolver software escrevendo em Português, o leitor de livro traduzido ainda tem que enfrentar os prováveis erros na tradução do código fonte.

Por tudo isso, e também pelo fato de o leitor desse livro certamente complementar seus conhecimentos lendo diretamente em Inglês, creio que o melhor que se deve fazer pelo profissional em fase de formação é ensiná-lo com códigos de exemplo basicamente em Inglês, como é feito nesse livro.

8. **Mais lenha na famosa “briga” entre C++ e java.** A linguagem C++ implementa os conceitos de orientação a objetos, que são de grande valor em projetos de tecnologia de informação. Outras linguagens também o fazem. A linguagem java também implementa muito bem os conceitos de orientação a objetos, e claramente concorre com C++ como alternativa tecnológica para implementação de sistemas. Há um debate quente sobre C++ versus java. Várias universidades acreditam que não é mais necessário ensinar C/C++, porque supostamente java a substituiria com vantagem de ser mais segura, elegante, robusta, portátil, etc. Eu sustento que embora a linguagem java tenha várias qualidades, é um erro parar de ensinar C/C++. Na seção “Java × C++”, na página 70, discute-se a comparação entre C++ e Java, do ponto de vista estrutural e estratégico.

Em pleno século 21, C++ é uma linguagem viva, vibrante e com muito, muito, futuro. Pensando assim, o mercado precisa seguir melhorando sempre o material tutorial para C/C++, inclusive porque há novidades acrescentadas a linguagem há relativamente pouco tempo, (e.g. namespace, RTTI, programação genérica com template, STL), e que precisam ser ensinadas com material didático sempre melhorado.

Enfim, a partir das justificativas acima, creio que ainda há trabalho a ser feito no campo de autoria de livros para C++. Trabalho diretamente lecionando C++, programação e desenvolvimento de sistemas usando o paradigma orientado a objetos. Percebo que há constante escassez de gente qualificada, seja para os diversos projetos que faço na universidade, seja para ofertas de emprego em empresas boas (isto é, onde se desenvolve software de “alta densidade”). Há

inúmeras ofertas profissionais para quem realmente conhece esse assunto, com remuneração em geral bem acima da média nacional.

Espero que a disponibilização desse livro sobre C++ incentive a formação de profissionais de software. Como efeito imediato, espero perceber um aumento na quantidade e qualidade de profissionais de software nessa linguagem. Assim, haverá gente para compor quadros especializados para tocar projetos interessantes que vão surgir. Como efeito indireto, e muito desejável, espero ajudar o Brasil, bem como todos os povos que falam Português, a melhorar sua qualificação técnica, e com isso obter melhoria no nível de emprego humano em atividade econômica. Em última análise, espero colaborar com o desenvolvimento econômico e bem estar social.

1.2 Histórico do livro

Esse é um livro que vem sendo escrito aos poucos. A seção abaixo mostra a evolução das diversas versões do texto.

1. Em 1992 foi feita uma versão chamada “Conceitos Básicos de C / C++”, para um curso oferecido à Rede Ferroviária Federal. Nessa época, estava na moda usar o compilador Borland C++ 3.1. (No período de 1993 a 1998, o autor esteve no Japão, Universidade de Chiba, fazendo curso de Doutorado).
2. Em 1999, foi feita uma versão profundamente modificada, chamada “Conceitos de software para sistemas de armas”, para um curso oferecido à Marinha do Brasil. Esse curso foi efetivamente “C++ e orientação a objetos, em console”. Para esse curso, usou-se o compilador Borland C++ 5.0.
3. Em 31 de agosto de 1999 foi feita a uma versão “esqueleto” desse livro, com o projeto de escreve-lo aos poucos e disponibilizar as versões correntes em página web. Definiu-se o nome do livro como “C/C++ Multiplataforma”. Chamou-se essa versão de 1.0, ainda com inconsistências. Mesmo assim foi tornada pública na Internet a pedido de alunos.
4. Em 15 de Fevereiro de 2000, a pedido de um amigo, eu publiquei a versão 1.1 do livro. Há inúmeras melhorias em relação à versão 1.0, mas essa versão ainda continha inconsistências.
5. Em 15 de Março de 2000, após várias modificações e acréscimos em relação a versão anterior, foi disponibilizada a versão 2.0. Essa versão ainda possuía diversas seções incompletas (marcadas com //todo), e algumas seções inconsistentes. Essa versão trouxe uma mudança de título, que passou a se chamar “C/C++ e Orientação a Objetos em Ambiente Multiplataforma”.

6. Em Julho de 2000, fiz diversas melhorias baseado na experiência de um curso oferecido com esse livro. Essa foi a versão 3.0.
 - a. Refeito e melhorado o capítulo “Conheça Seu Compilador”.
7. Em Fevereiro de 2001, após um curso para o PCT-DEL, surgiram diversas sugestões de melhorias, que foram implementadas nessa versão 4.0.
 - a. Modificou-se bastante o capítulo 1: “Introdução”, inclusive com o acréscimo de seções sobre estratégia na escolha de tecnologia. Para escrever essa parte, aproveitei material do interessante curso de MBA em “Inteligência Competitiva e Gestão do Conhecimento”, recém concluído.
 - b. Acrescentou-se capítulo de boa programação vs. má programação
 - c. Acrescentou-se capítulo sobre STL
 - d. Acrescentou-se tutorial para DLL em Visual C.
 - e. No capítulo “Conheça o Seu Compilador”, acrescentou-se referência para o compilador Borland Builder 5.0.
 - f. Acrescentou-se índice remissivo. Nas próximas versões, esse índice deve melhorar.
 - g. Acrescentou-se uma seção inicial de introdução dando ao leitor uma justificativa para existência desse livro, face ao ambiente do mercado de tecnologia de informação.
 - h. Revisão geral.
8. Em Agosto de 2001, versão 5.0 (a versão 5.1 é basicamente idêntica a versão 5.0, apenas acrescentando-se os links PDF). Essa foi a primeira versão divulgada amplamente pela web.
 - a. Incluiu-se um índice.
 - b. Retirados quase todos os trechos de código c++ em português do livro. Dessa forma, ilustra-se que é boa programação escrever o código em si em inglês.
 - c. Incluiu-se informação sobre uso da ferramenta libtool, para criação de bibliotecas de ligação dinâmica em unix (embora ainda seja preciso melhorar essa informação).
 - d. Incluiu-se informação sobre desenvolvimento de pacotes RPM para linux.
 - e. Melhorou-se o capítulo sobre tratamento de exceção.
 - f. Melhorou-se o capítulo sobre namespace.

- g. Acrescentou-se informação sobre construtor de cópia, e criação de objetos com memória alocada.
 - h. Acrescentou-se um capítulo sobre dicas de programação.
 - i. Acrescentou-se mensagem do patrocinador, e com isso iniciou-se o formato compatível com o modelo de negócios atual do livro.
9. Em Março de 2002, versão 6.0. (essa versão não chegou a ser divulgada)
- a. Inclui-se diversos comentários construtivos e sugestões de leitores enviadas por email.
 - b. Re-escreveu-se toda a parte de estratégia e C++, o que corresponde a primeira parte do livro.
 - c. A Bibliografia foi toda refeita, com inclusão de inúmeros livros e sites na web. Nessa parte há um breve comentário sobre cada uma das referências.
 - d. Nova divisão do livro em 8 partes.
 - e. Refez-se os slides que apoiam o curso de C++, em concordância com as 8 partes do livro.
 - f. Inclusão de capítulo sobre programação cgi com VBMcgi.
 - g. Incluiu-se uma seção com um componente para varrer árvore de diretórios em unix.
 - h. Acrescentou-se o compilador Dev-C++ na lista de compiladores comentados no capítulo “Conheça o Seu Compilador”.
 - i. Incluiu-se exercícios.
10. Em Março de 2006, versão 7.0.
- a. Reformulação profunda nos slides que acompanham o curso.
 - b. Melhorias na parte de STL.
 - c. Mudança na ordem das partes do livro.
 - d. Pequenas mudanças gerais

1.3 Como usar o livro

O livro está dividido em 8 partes. Abaixo está o resumo do conteúdo de cada uma das partes.

Parte 1: Porque C / C++ ? Tecnologia & estratégia – página 23. Nessa parte discute-se possibilidades e estratégias para profissionais e empresas que atuam no mercado de tecnologia de informação. No contexto da discussão, aborda-se a escolha de tecnologia como uma escolha estratégica. Discute-se também

aspectos da sociedade do conhecimento. Essa parte do livro não explora a “carne” da linguagem C/C++ em si. Para leitores interessados em apenas aprender a linguagem, essa parte pode ser ignorada. Pode-se tomar essa parte do livro como uma resposta a seguinte pergunta: “ Por que estudar C++ ao invés das alternativas ? ”.

Parte 2: Fundamentos de C / C++. Nessa parte discute-se os fundamentos da linguagem. Trata-se de um texto dirigido a iniciantes, mas que eventualmente pode ser útil também para profissionais experientes. Destaca-se o capítulo “Conheça o Seu Compilador”, que mostra o uso de vários compiladores C++ diferentes.

Parte 3: C++ e Orientação a objetos. Nessa parte apresenta-se os conceitos básicos de orientação a objetos (herança, encapsulamento, polimorfismo, etc.), e discute-se a forma de implementar esses conceitos em C++.

Parte 4: C++ e Orientação a objetos (adendo). Nessa parte são discutidos assuntos complementares da programação orientada a objetos. Por exemplo: Tratamento de exceção (try-catch), namespace, RTTI (Run Time Type Identification), uso da biblioteca padrão de C++ (inclui acesso a disco), manipulação segura de strings, uso de “const”, etc.

Parte 5: Programação para web com C++ e VBMcgi. Nessa parte discute-se como utilizar na prática a linguagem C++ no desenvolvimento de sistemas para web (programação CGI). Para apoio ao desenvolvimento para web com C++, usa-se a biblioteca multiplataforma gratuita VBMcgi (www.vbmcgi.org), mantida pelo autor. Nessa parte há também uma discussão estratégica sobre as alternativas tecnológicas para programação cgi, vantagens e desvantagens das opções.

Parte 6: Programação genérica (template). Nessa parte discute-se programação genérica, que em C++ é feita com o uso da palavra reservada “template” (fôrma, gabarito).

Parte 7: STL – Standard Template Library. Nessa parte discute-se de forma introdutória o uso da poderosa biblioteca STL, que é toda escrita em genérico, e implementa diversos contenedores e algoritmos muito usados em diversas aplicações. STL é atualmente considerado como parte da biblioteca padrão de C++.

Parte 8: Complementos. Nessa parte apresentam-se alguns conceitos de fechamento, incluindo incompatibilidades entre compiladores, dicas, componentes de programação, etc.

C++ Multiplataforma e Orientação a Objetos

Parte 1:

Porque C++ ?

Tecnologia & Estratégia

2 Tecnologia da informação e negócios

Há inúmeras linguagens de computador.

Porque estudar a linguagem C++ ao invés alguma das alternativas ?

Esse capítulo procura responder a essa pergunta. A resposta é um pouco demorada, pois inclui discussão sobre negócios (*business*). Mesmo para quem está ávido por aprender a programar, é uma leitura que vale a pena.

2.1 Experiência de vida e visão de negócios

Em 1982, tendo recém concluído o curso técnico de eletrônica, eu programava em FORTRAN, assembler de 8080 (antiga CPU da Intel) e calculadoras HP e Texas. O primeiro emprego que tive na vida foi com instalação e manutenção de equipamentos para estúdio de TV, na TV Record do Rio de Janeiro, onde tive a oportunidade de ver como um estúdio de televisão funciona por dentro. Um dos legados mais importantes que conservo do período de 1 ano e meio que trabalhei lá é a interpretação que passei a dar a programas de TV. Quem já acompanhou a produção de programas de TV, assiste aos programas pensando em como atuam os profissionais do estúdio e nos recursos técnicos que se dispõe. O conteúdo do programa em si é apenas uma das informações que se observa. Nunca mais assisti TV da mesma forma.

Posteriormente, a partir de 1985, trabalhei como técnico num projeto de pesquisa em controle de processos químicos, no programa de engenharia química da COPPE. Esse trabalho técnico tornou-se trabalho de engenharia, com projeto de software e hardware para aquisição e exteriorização de sinais analógicos. Na sequência, o mesmo trabalho tornou-se uma pesquisa de mestrado, em que se analisou, pesquisou, implementou e testou algoritmos de controle multivariável, aplicados experimentalmente em tempo real numa planta piloto. Para a implementação desse projeto de pesquisa, foi necessário um grande estudo de programação e desenvolvimento de sistemas. O que se estudava no curso de engenharia e de mestrado, aplicava-se no projeto de pesquisa. Desenvolvi um programa enorme que implementava o controle em tempo real, exibia graficamente os resultados, recebia comandos do console (implementando multi-task cooperativo), interagiu com o hardware de aquisição e exteriorização de dados. Esse programa foi desenvolvido em C++ (compilador Borland C) e executado num PC-XT (o computador que se usava na época), com sistema operacional MS-DOS 3.3. O programa oferecia ainda um ambiente para que terceiros pesquisadores acrescentassem código de algoritmo de controle que desejassem testar. Em 1991 concluí a tese de mestrado intitulada “ambiente de estudos para controle de processos”, resultado de mais de 6 anos

de trabalho. O desenvolvimento desse trabalho ajudou a sedimentar diversos conceitos de programação tais como programação em tempo real, interface com hardware, gestão de projeto de software, reutilização de código, especificação de ambiente de software, programação orientada a objetos, etc.

Entre outros legados dessa experiência, quando vejo uma placa de hardware (por exemplo placa com conversor AD/DA, ou placa de vídeo, ou de som) com um manual para desenvolvimento de software, imagino sempre o trabalho do engenheiro que desenvolveu todo o kit. Nunca mais consumi um kit desses da mesma forma. Adicionalmente, costumo imaginar as dificuldades por que passa um gestor de projeto de software que é desenvolvido por uma equipe. É fácil criticar os bugs dos programas que usamos, mas gerir uma equipe para desenvolver um programa grande e sem bugs não é tarefa fácil.

Quando fui fazer doutorado no Japão, a partir de 1993, dediquei-me a pesquisa científica na área de controle. Boa parte do trabalho consistia de ler, discutir e entender diversos *papers* (artigos científicos), além de livros e outras referências. Para implementar os testes da pesquisa, era preciso desenvolver muitos programas, em C++ e em Matlab. Por motivos indiretos foi necessário um estudo enorme de programação e desenvolvimento de sistemas. Entender o vasto material que se estava estudando já era tarefa árdua, mas não o suficiente. O trabalho de doutorado essencialmente é o de produzir conhecimento científico, em geral propondo um método, e publicar as conclusões na forma de *paper*, numa revista técnica indexada. Com isso, empurra-se a ciência para frente. A publicação é parte indissociável do trabalho científico. Não pode haver uma “ciência oculta”. Ciência por definição não pode ser oculta. É preciso que se publique as conclusões do trabalho, para que uma “comunidade científica” possa critica-lo. Qualquer conclusão necessariamente deve poder ser confirmada por fatos. Um trabalho científico publicado (tornado público) tem dois destinos possíveis: ou é contestado (também publicamente), ou por exclusão é aceito como correto. Por passar pelo curso de doutorado, aprende-se muitíssimo, publica-se alguns *papers* e observa-se o ambiente no qual se produz conhecimento e se expande a ciência. Depois de passar por esse processo, nunca mais interpretei a ciência como antes. Um *paper* é antes de tudo a síntese de um projeto de pesquisa de um ser humano.

Depois do doutorado, achei que estava faltando visão de negócios na minha formação, e no ano 2000 fiz o curso de MBKM - um mestrado *lato sensu* em administração de negócios com ênfase em inteligência empresarial e gestão do conhecimento. Em paralelo com esse mestrado, dediquei-me a diversos cursos, experiências e pesquisas sobre programação e desenvolvimento de sistemas para web, banco de dados, análise de sistemas, programação orientada a objetos e outros assuntos relacionados a software. Durante o um de duração ano desse mestrado, ocorreu a última fase de euforia da web, e o “estouro da bolha”. Tudo o que estava ocorrendo no cenário econômico foi tema de debates e análises no

curso MBKM. Apoiado pelos estudos desse curso, que incluíram diversos estudos de caso de negócios, passei a ver os negócios de forma diferente. Agora, quando sou cliente de uma empresa, reflito sobre a missão da empresa, seu modelo de negócios, sua estratégia e toda a dinâmica pela qual a empresa passa para por fim oferecer o tal produto. Após esses estudos, nunca mais fui um cliente de empresa como antes.

Com respeito a atividades acadêmicas no DEL (Departamento de Engenharia Eletrônica e Computação da UFRJ), venho ultimamente dedicando-me a lecionar linguagens de programação, com ênfase em C++ e orientação a objetos.

Adicionalmente, ainda após o doutorado, tive oportunidade de executar algumas consultorias em que se desenvolveu software em parceria com empresas privadas. Uma dessas experiências foi particularmente boa. O software em questão era bastante avançado. Um módulo foi o desenvolvimento de um applet java com várias funcionalidades, sendo uma delas a de enviar dados criptografados (desenvolvido em cima de soquete seguro SSL) para um serviço para Windows NT, escrito em C++. Não se encontra gente facilmente para desenvolver esse tipo de aplicação. Foi de particular valor a experiência de colaborar com a equipe da empresa privada, num contexto onde há necessidade de respeito a prazos, discussão de opções tecnológicas, qualidade de software, necessidade de integrar os módulos, prestar atenção as especificações exigidas pelo cliente, etc.

Essas experiências de vida acumuladas mostram claramente que existe uma dinâmica econômica e social sobre a qual as empresas e os negócios se desenvolvem. A visão de negócios claramente influencia as atividades profissionais que fiz – televisão, processos químicos, instrumentação, controle, ciência e consultoria em tecnologia de informação. Obviamente, essa mesma visão influencia muitas outras atividades profissionais, praticamente todas as que existem.

Percebi que na compreensão e gestão do mundo, há relativamente poucos níveis pensamento que se situam acima do nível de negócios. Questões relacionadas a princípios morais, existenciais, filosóficos, preservação ambiental e liberdade são alguns desses níveis de pensamento. Mas o que é importante enfatizar é que a tecnologia (particularmente a tecnologia de informação) é claramente um nível que situa-se abaixo do nível de negócios. Em outras palavras, em geral a TI serve aos propósitos dos negócios, e não o contrário – seja do ponto de vista de quem usa a tecnologia, seja do ponto de vista de quem a integra, seja do ponto de vista de quem a produz. Na figura 1, mostra-se os níveis de pensamento humano, mostrando como os negócios situam-se acima da tecnologia de informação.

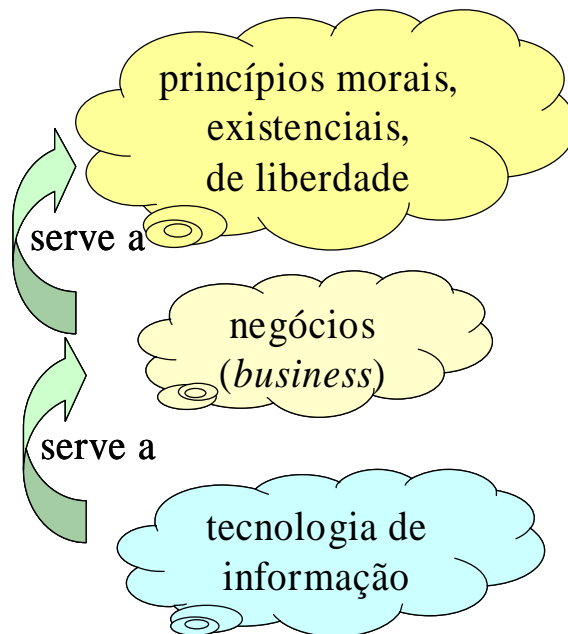


Figura 1: Níveis de pensamento humano

Porque discutir negócios num livro sobre C++ ? Resposta: porque por entender de negócios, pode-se avaliar melhor a estratégia das empresas que produzem as ferramentas de TI que se usa. Em função dessa estratégia, é possível interpretar as novas funcionalidades que as ferramentas tem, e avaliar se é ou não interessante estudá-las e usá-las. Quem entende de negócios nunca mais consome ferramentas de TI como antes. Nas seções seguintes, a discussão prossegue.

2.2 Negócios forjam a tecnologia

As forças que impulsionam o desenvolvimento de produtos e o ambiente profissional de tecnologia de informação são fortemente influenciadas pela visão de negócios. Portanto, é muito importante que um profissional ou estudante de tecnologia de informação seja minimamente iniciado em negócios, administração e assuntos relacionados. A partir de uma compreensão mínima de como geralmente se pensa e age no ambiente de negócios, pode-se definir melhor uma estratégia de atuação pessoal e/ou empresarial. É útil entender de negócios também pelo fato de que a experiência mostra que uma carreira no setor de TI muitas vezes culmina com o profissional tornando-se gerente ou empresário. Por um motivo ou por outro, o conhecimento de negócios é de grande valor. Como se sabe, o empresário deve entender bem de negócios, ou sua empresa provavelmente quebrará. O cargo de gerente igualmente requer do profissional visão de negócios. É comum que o gerente tenha remuneração variável, e existe necessidade de ele assumir e compartilhar riscos praticamente como se fosse o próprio dono da empresa. Somente consegue esse posto profissional a partir de uma “atitude empreendedora”, ou “visão de negócios”.

Mesmo enquanto não se atinge posto de gerência ou de dono de empresa, um profissional de TI é mais valorizado por compreender o significado de negócios em que se insere o projeto com o qual está envolvido. Por compreender bem a missão da empresa que representa, o profissional terá melhores condições de executar adequadamente suas tarefas. Boa parte dos clientes de TI são empresas que compram sistemas com objetivo de melhorar a eficiência de seus próprios negócios. Por exemplo: uma loja compra um sistema para automatizar as vendas e fazer gestão integrada do estoque. Para que se possa entender claramente o que o cliente quer – seja para desenvolver ou para instalar um sistema, ou ainda para treinar os operadores – é preciso ter visão de negócios. Quem paga pelo sistema, em última análise é o dono da empresa, e é preciso que ele seja convencido – na sua linguagem – de que o sistema tem valor.

O sistema de informação deve ser desenvolvido para satisfazer em primeiro lugar o dono da empresa, e não o operador ou o gerente de TI.

Há inúmeras referências de boa qualidade para quem se interessa por negócios. Revistas Brasileiras como Exame, Isto É Dinheiro, e estrangeiras tais como Fortune, The Economist e Fast Company divulgam excelentes matérias sobre negócios. Todas elas possuem também página na web (veja a seção de bibliografia para referências, na página 433).

Uma referência boa para o estudar o relacionamento entre tecnologia e negócios é o livro “Empresa na velocidade do pensamento”, por Bill Gates [42]. Esse livro não é uma propaganda de Windows. Trata-se de um texto MUITO acima desse nível, abordando o funcionamento de diversos negócios, e a forma como devem ser os sistemas de informação para suprir suas necessidades. O livro descreve claramente como um sistema de informação deve ser projetado para gerar valor para diversos tipos de negócios. Projeto de aviões, banco e financeira, venda de veículos, educação, saúde, gestão e biotecnologia são alguns dos tipos de negócios que se aborda. A linguagem utilizada enfatiza os resultados e a relação custo/benefício que a tecnologia traz (ou deveria trazer) para os negócios, que em última análise são os clientes da tecnologia. Uma das idéias sustentadas nesse livro é a frase abaixo.

Se a velocidade de circulação da informação é rápida o suficiente, o próprio significado dos negócios (missão da empresa) precisa ser repensado.

Com a Internet/web, a informação circula quase instantaneamente. Como devem ser os negócios nesse ambiente ? Como devem ser os sistemas de informação

para atender a essas necessidades ? Que oportunidades se abrirão para profissionais e empresas de TI ? Que estratégia esses profissionais e empresas devem ter para aproveitar bem essas oportunidades ?

Outra referência muito boa é o filme “Piratas da Informática” (*pirates of silicon valley*). A versão em VHS ou DVD desse filme está disponível em boas locadoras. O enredo conta a história verídica do início da era dos computadores pessoais, desde o início da década de 1970 (quando não haviam computadores pessoais) até a polêmica compra de ações da Apple pela Microsoft. Conceitos como “capitalismo de risco” (*venture capitalism*), “empreendedorismo” (*entrepreneurship*), “inteligência competitiva”, “inovação”, “competência essencial” (*core competence*) são alguns dos que se usa para descrever as cenas do filme. Entre os personagens, incluem-se Steve Jobs, Steve Wozniak (sócios iniciais da Apple), Bill Gates, Paul Allen e Steve Balmer, (sócios iniciais da Microsoft). Há cenas gloriosas, como aquela em que a Microsoft consegue tornar-se fornecedora do sistema operacional DOS para a IBM acrescentar no seu computador pessoal IBM-PC prestes a ser lançado.

2.3 Classificação dos atores do ambiente para TI

Uma forma de se resumir o ambiente econômico relacionado a tecnologia de informação é classificar os seus atores em 3 categorias, como mostrado abaixo.

1. Consumidores de TI
2. Integradores de solução
3. Produtores de tecnologia

Os consumidores de TI são lojas, secretarias, repartições, escritórios, escolas, comércio, indústria, etc. São entidades que por sua própria atividade precisam de sistemas de informação para apoiar o trabalho administrativo, contabilidade, pesquisa de informação, etc. O objetivo dos atores dessa categoria é usar a tecnologia de informação para obter produtividade ou algum outro valor na missão de suas respectivas entidades.

Os integradores de solução são empresas que atendem os consumidores de TI como clientes. Uma atividade característica dessa categoria é a análise de sistemas. O analista estuda o problema do cliente e faz um modelo para a sua solução. Em seguida, projeta o sistema e envia informações para que o sistema seja desenvolvido por programadores. Há inúmeras empresas nesse segmento, por exemplo Alternex, Data-Bras e Tera-Brasil e Módulo, apenas para citar algumas empresas brasileiras. Além da excelência tecnológica, empresas desse segmento cultivam valores importantes no que tange a relacionamentos no mercado. Por isso a tradição e a qualidade do atendimento são fundamentais. Outra vertente importante para empresas desse segmento é a contínua atualização tecnológica e a certificação dos profissionais.

Os produtores de tecnologia são empresas ou entidades que desenvolvem produtos de aplicação geral para TI. Dentre os atores nesse segmento encontram-se Microsoft, Oracle, Sun, Macromedia, Adobe, Rational, Red Hat, Apache, Gnu, Conectiva, Módulo e Microsiga. Há vários programas de computador que são desenvolvidos para aplicação genérica. Por exemplo: sistema operacional, editor de texto, banco de dados, cliente web, servidor web, compilador C++ e ferramenta de autoria de conteúdo web. Quem desenvolve esse tipo de produto tem como cliente um público diversificado, tanto na categoria de consumidor de TI como na categoria de integrador de solução.

2.4 Ambiente econômico e modelo de negócio

Existe um “ambiente econômico”. Resumidamente, há produtos e serviços em oferta, e um mercado para consumi-los. A oferta é suprida por indivíduos ou grupos que possuem empresas, e estão a frente de negócios. Há também empresas públicas, isto é, que pertencem a sociedades. A história mostra que empresas são criadas, e eventualmente morrem. Se o ambiente muda, uma certa demanda pode desaparecer, e com isso uma atividade econômica pode deixar de fazer sentido. Portanto, a empresa que tem por missão suprir essa demanda tende a quebrar ou a mudar de missão.

Há exemplos de sobra de empresas que quebraram ou foram forçadas a adaptar-se porque o ambiente mudou, e a demanda original deixou de existir. Segue-se apenas um exemplo: muito antigamente não havia qualquer iluminação nas ruas. A primeira tecnologia que se usou para suprir a demanda de iluminação pública foi baseada em gás. Surgiu portanto demanda para equipamentos de iluminação baseado em gás. A tecnologia de eletricidade para iluminação pública tornou obsoleto o uso de gás para iluminação pública. A partir de um certo momento, deixou-se definitivamente de se fabricar equipamentos para iluminação a gás. Uma empresa que vivesse de fabricar esse tipo de equipamento estava fadada a fechar ou mudar de missão, não por suas deficiências internas, mas pela mudança do ambiente econômico.

No passado, a velocidade com que as novidades surgiam e eram disseminadas era bem menor que atualmente. O ambiente econômico mudava devagar. Em muitos casos, era preciso que uma geração inteira fosse substituída pela nova geração, com novas idéias, para que inovações fossem incorporadas ao uso social. Atualmente, a velocidade de mudança do ambiente é tão grande – particularmente em TI – que a possibilidade de que seja necessária a mudança de missão da empresa, ou que o profissional precise reciclar-se é praticamente uma certeza. Esse é um ambiente “de alta voltagem”, que traz muitos riscos, mas também muitas oportunidades. É muito recomendável que indivíduos e empresas pensem de forma estratégica para atuar no ambiente econômico em constante mutação.

2.5 Web impacta ambiente econômico

*A Internet/web não está criando somente novos negócios.
Ela está criando novos modelos de negócio.*

A frase acima foi repetida em diversas reportagens e artigos que tentavam entender como fica o ambiente econômico com a existência de Internet e web. Um fator que produza oportunidade para novos negócios já é por si só de muita importância. Quando um fator chega a produzir novos modelos sobre os quais se concebe negócios, então definitivamente trata-se de fator de importância capital. Esse é o caso da Internet e da web (um subconjunto da Internet). Algumas atividades que estão presentes no ambiente econômico pré-Internet eventualmente poderão deixar de fazer sentido. Novas atividades surgem. Quais são as oportunidades que estão se forjando nesse ambiente ? Que estratégia indivíduos e empresas devem adotar para posicionar-se no sentido de aproveitar bem as oportunidades do ambiente que está se formando ? Esse tipo de pergunta ecoa na cabeça de muita gente. Em Abril de 2002, o canal de TV BBC World exibiu uma série de programas intitulados *The future just happened* (o futuro acabou de acontecer), em que mostra com vários exemplos concretos de que forma a Internet/web está afetando a forma como as pessoas se relacionam a nível pessoal e profissional, e como novos modelos de negócio estão silenciosos e rapidamente se implantando. Enfim, como a humanidade está escrevendo sua história no novo ambiente que já chegou. O profissional de TI é diretamente interessado nessa discussão, pois suas oportunidades profissionais dependem disso.

No final do segundo milênio houve um estado de euforia coletiva com as potencialidades de ganhar dinheiro a partir de novos negócios, viabilizados a partir do novo ambiente com Internet/web. A euforia terminou (ou diminuiu muito) quando as ações das novas empresas começaram a cair. Foram testados modelos de negócio absurdos, e muitos se mostraram inviáveis. Mesmo antes da queda das ações, diversos artigos mostraram que havia algo de fundamentalmente errado e insustentável com alguns dos pré-supostos que alimentavam a euforia. Um desses pré-supostos era que “quanto mais tráfego numa página web, melhor, pois ganha-se dinheiro com propaganda”. Se a lógica fosse apenas essa, teoricamente o seguinte modelo absurdo de negócios seria viável. Uma página chamada “dinheiro_de_graça.com” dá dinheiro para os visitantes, sem sorteio. A atratividade da página obviamente seria enorme. Se a página fatura proporcionalmente ao tráfego supõe-se que faturaria muito nesse caso. Portanto, supostamente haveria dinheiro para pagar aos visitantes e ainda restaria um bom lucro. A fragilidade do modelo é que o tráfego alto de uma página web somente é valorizado como veículo de propaganda caso corresponda

em capacidade real de influenciar um público consumidor para o qual se fará a propaganda. A atratividade aleatória que se geraria com um site tipo “dinheiro de graça” não seria facilmente aproveitável para uma campanha publicitária. Portanto não seria um bom negócio fazer propaganda nesse tipo de site da web.

O fato de que muitos modelos de negócio testados se mostraram falhos não quer dizer que a importância da Internet/web para o mundo seja pequena. Como subproduto da “euforia de Internet” houve um desenvolvimento tecnológico sólido, e a tecnologia de web tornou-se bastante testada e madura. Apesar do exagero inicial, ainda há grandes oportunidades para aplicação de tecnologia web. Um nicho onde atualmente há particularmente muita oportunidade de aplicação de tecnologia web são os sistemas corporativos, incluindo automação comercial, industrial e de escritórios. Com criatividade, conhecimento e visão de negócios, pode-se ainda identificar muitos nichos a serem atendidos.

2.6 Analogia entre militarismo e negócios

Há analogias possíveis entre o militarismo e o mundo dos negócios. **O general (comandante máximo das forças militares) vê o campo de batalha como o executivo (chefe/comandante máximo da empresa) vê o mercado.** Ambos querem conquista-lo. É preciso defender o pedaço que já se conquistou, e ter uma estratégia para conquistar o pedaço restante. Em tese, ambos atuam dentro da lei, embora usando meios diferentes. Ambos tem tentação de agir fora da lei para atingir seu objetivo. Em ambos os casos, há mecanismos que procuram inibir a atuação fora da lei.

A terminologia usada em ambos os casos também possui muito em comum. Quando um inimigo invade seu território, o general refere-se ao fato como uma “agressão”. É muito comum um executivo usar o mesmo termo “agressão” para referir-se a uma investida de um concorrente que lhe toma uma parte do mercado.

Tanto o general quanto o executivo precisam de estratégia. De fato, uma das definições da palavra estratégia é: “estratégia é a arte do general”. Mais especificamente, pode-se definir estratégia como sendo “o plano de ação que se adota com a intenção de chegar a um objetivo”. “Tática” é uma estratégia de curto prazo.

Para se ter “pensamento estratégico” pressupõe-se a existência de objetivos.

A falta de visão estratégica pode levar a uma situação como a da Alice no País das Maravilhas. A Alice pergunta ao gato mágico: “Que caminho devo tomar?”. O gato responde: “Depende. para onde você quer ir?”. Alice novamente: “Não sei”. E o gato: “Então qualquer caminho serve”.

Não há vento favorável para quem não sabe para onde quer ir.

Tanto o general quanto o executivo cultivam objetivos de longo prazo. Ambos em geral dependem fortemente de tecnologia, e portanto preservam e gerenciam os conhecimentos tecnológicos estratégicos para garantir seus objetivos de longo prazo. Ambos consideram o estabelecimento de alianças para facilitar a obtenção de seus objetivos.

2.6.1 Inteligência

A “inteligência militar” é uma atividade desenvolvida por espões (o cargo pode ter outro nome) em função de demandas criadas por guerras. Essa atividade inclui a coleta sistemática de informações, e sua classificação. Há uma metodologia própria para essa atividade. Algumas profissões usam métodos semelhantes aos usados pelos espões. Por exemplo: bibliotecário ou documentarista. Após a segunda guerra mundial, muitos espões ficaram sem emprego. Gradualmente vários deles passaram a usar seus conhecimentos de coleta e classificação de informações para auxiliar negócios. Essa atividade passou a ser chamada de “inteligência competitiva”.

Uma definição para *inteligência competitiva* é: “o processo de coleta e classificação sistemática de informações sobre as atividades dos concorrentes e tendências gerais do cenário econômico, executado dentro dos limites da lei.” A inteligência competitiva não é espionagem ilegal, não é um processo ilógico especular sobre o futuro (e.g. astrologia), não é um simples relatório gerado por um software. É um processo sistemático de coleta e interpretação de informações com objetivo de evitar surpresas, identificar tendências, ameaças e oportunidades, propor estratégias para obtenção de vantagem competitiva. Boa parte do trabalho de inteligência competitiva é o de coletar informações em órgãos abertos (em geral informação diferente daquela obtida gratuitamente da Internet, mas informação de empresas respeitáveis de comunicação), e interpreta-las em favor dos objetivos da entidade que se representa.

Seja por exemplo uma empresa que produz agrotóxico e/ou alimento transgênico. Um parâmetro importante para essa empresa é monitorar o grau de aceitação dos seus produtos pelas diversas sociedades do mundo. Caso um certo país esteja demonstrando claramente intolerância para o uso desses produtos, provavelmente alguma ação precisará ser tomada pela diretoria da empresa. Se um país funda uma “sociedade para repudiar alimentos transgênicos” (ou coisa parecida), algum jornal provavelmente publicará matéria sobre o assunto.

Outro exemplo: se está ocorrendo um grande aumento de demanda em cursos de utilização de software gratuito (informação que se pega de jornal), pode-se concluir que as oportunidades futuras no mercado de TI provavelmente

requerirão experiência com essa tecnologia. Portanto uma proposta de estratégia é iniciar uma linha de trabalho usando software gratuito, e com isso cultivar a cultura de usa-lo.

2.7 Conhecimento empacotado

Seja o termo “conhecimento empacotado” (*packed knowledge*), que refere-se a uma condição de mercado de um produto ou serviço. Há muitos produtos e serviços disponíveis. Em muitos deles, há componente intensiva de conhecimento. Mas muitas vezes o mercado apenas vende o resultado do conhecimento, e não o conhecimento em si. Isto é: trata-se de um “conhecimento empacotado”. Compra-se o conhecimento, e pode-se usa-lo, mas não se pode “abrir o pacote”. As características da condição de mercado de um produto ou serviço ser classificado como “conhecimento empacotado” são:

- O seu custo deve-se em grande parte à remuneração do conhecimento necessário a produção, e não ao material físico que compõe o produto.
- O ato de compra não transfere para o comprador o conhecimento de como se produz, mas apenas o direito de uso e o conhecimento de como se usa.
- O vendedor procura preservar-se por proteger o seu conhecimento. Isso é feito seja por meios legais (copyright, patente, etc.), seja por dificultar a tentativa do comprador de apoderar-se do conhecimento ou duplicá-lo sem interferência do vendedor (chave de hardware, segredo industrial, bloqueadores de engenharia reversa, etc.)
- O modelo de negócios típico leva o comprador à dependência tecnológica, e portanto econômica, em relação ao vendedor. Em outras palavras: quem compra um “conhecimento empacotado” tende a seguir comprando-o. Por engajar-se no modelo de negócios proposto pelo vendedor, o comprador não adquire o conhecimento para produzir o produto ou serviço em questão.
- Não raro o comprador investe tempo para aprender a usar o produto, devido a sua complexidade. Com intenção de preservar seu investimento de tempo no aprendizado, o comprador considera com preferência a compra do mesmo produto novamente, mesmo havendo alternativas e concorrentes. Essa característica gera efeito econômico de haver vantagem para o “primeiro a se mover” (*first mover*). Aquele que consegue mover-se antes dos concorrentes e estabelecer seu produto num segmento de mercado ainda virgem, terá vantagem adicional para seguir explorando esse segmento do mercado.
- O custo do desenvolvimento é em geral vultoso, envolvendo a remuneração de profissionais caros. Para obter retorno do investimento faz sentido econômico vender em escala global.

- Se o produto é muito usado, adquire mais valor.

O software é um exemplo perfeito de conhecimento empacotado. Quando compra-se um software, se está comprando o direito de executar um arquivo binário, e não o conhecimento sobre como fazer o tal programa. Em geral, não se adquire o fonte de um programa ao compra-lo.

Como referência, vale citar alguns outros exemplos de conhecimento empacotados. Não seria difícil citar ainda inúmeros outros.

- Boa parte do milho que se come hoje (seja diretamente, seja indiretamente quando usado como ração de animais) é o chamado “milho híbrido”. Por motivos genéticos, as qualidades que se espera do milho (crescer rápido, com espiga grande e de bom valor nutricional, etc.) são obtidas quando se cruza uma espécie de milho A com a espécie B, gerando 100% de milho híbrido AB. As sementes do milho híbrido são férteis, mas apenas 50% delas são do tipo AB. A partir da terceira geração, a proporção de milho AB é ainda menor. Ou seja, no processo de produção de milho, há a empresa produtora de sementes de milho híbrido, que vende sementes 100% AB, e o fazendeiro que executa o ato de plantar. A cada safra, o fazendeiro precisa comprar novamente as sementes. A empresa de sementes não fornece o milho A e o milho B, mas apenas as sementes AB prontas para plantar. O valor de se comprar essas sementes está no fato de que o milho “melhorado geneticamente” (supostamente) rende mais para o produtor.
- Sem exagero, podemos dizer que somos a civilização do petróleo. Dependemos do uso regular de petróleo em grandes quantidades, por inúmeros motivos. A indústria petroquímica alimenta diversas outras indústrias importantes. Para a indústria petroquímica funcionar, usa-se “catalisadores” (substâncias que auxiliam o processamento químico do petróleo). Sem os catalisadores, a indústria petroquímica não funciona. E a vida útil de catalisadores não é infinita. Mas os fornecedores não vendem o conhecimento para se fazer o catalisador. Vendem apenas o catalisador pronto para usar.

Num mundo já muito complexo, e ainda com complexidade crescente rapidamente, é impossível deter todo o conhecimento num único povo. De fato, a idéia de produzir tudo que é necessário dentro do país é considerada por muitos como uma estratégia já impossível de se seguir. Muitos produtos importantes são hoje produzidos numa parceria econômica em que participam empresas em vários países. O computador que o leitor usa possui componentes físicos e software fabricados/desenvolvidos em diversos países. Nenhum país ou empresa isoladamente é capaz de fazer do zero um computador, e vende-lo no preço que se está praticando hoje em dia. Num exemplo brasileiro, o avião da Embraer é produzido numa inteligente parceria de diversas empresas,

coordenada pela Embraer. Não faz sentido pretender fazer 100% do avião dentro da empresa. Por vários motivos faz mais sentido coordenar a cooperação de várias empresas especializadas em componentes do avião.

Numa situação de complexidade, é particularmente fácil que produtos tipo “conhecimento empacotado” se estabeleçam. A competição muito severa e o ambiente em rápida mutação faz com que a toda hora os modelos de negócios precisem ser repensados. Quando uma empresa consegue definir uma fórmula de modelo de negócio em que sua atividade faz sentido, isso já é um grande feito. Nessa situação, em geral não se questiona o preço de suprimentos. Por exemplo: seja uma empresa de telefonia celular. Essa empresa precisa de um sistema de informação para calcular a tarifação dos clientes. Suponha-se que um integrador de solução já forneceu um sistema, que é uma solução satisfatória. Na hora de decidir por expansão do sistema, o gerente de tecnologia tenderá a escolher a mesma solução, pois já há cultura na empresa de uso do sistema. Por motivo semelhante, há tendência na padronização de sistema operacional, ferramenta de uso escritório (pacote *office*), etc. A complexidade grande do ambiente é fator de inibição da busca de alternativas ao que está funcionando, pois o aprendizado é em geral um investimento demorado e caro. Quando já se possui o conhecimento de como usar um produto, pode ser mais barato e inteligente compra-lo novamente do que investir para aprender a usar uma alternativa que custa mais barato para ser adquirida. Analisando o mesmo problema numa perspectiva de longo prazo, pode-se concluir pelo oposto (isto é, a longo prazo, pode se mais interessante investir para aprender a usar uma solução que é mais barata para ser adquirida).

2.8 O produto “software”

Voltemos ao software, que é um produto com características especiais, diferentes de quase todos os outros produtos, e um conhecimento empacotado por excelência. Um produto tangível (televisão, computador, etc.) precisa que cada unidade seja produzida para que seja vendida. De forma semelhante, serviço tangível (limpeza, preparação de comida, segurança, etc.) precisa ser feito a cada vez que é vendido. Já um software é um produto que na prática custa apenas para ser desenvolvido. O custo de produção ou cópia é tão pequeno que pode ser considerado nulo. Uma vez desenvolvido, a firma poderia em princípio copiar e vender o software indefinidamente, com excelente lucro. Mas nem tudo são flores para uma firma de software. A concorrência é muito grande, e não se sabe a princípio se um software vai ter boa aceitação no mercado. Portanto, o negócio de desenvolver software é um tanto arriscado. Além disso, o ambiente tecnológico muda rapidamente. Para desenvolver um software, é preciso investir em salários de profissionais caros, e gastar tempo. Quando o produto fica pronto, é preciso convencer o mercado a aceitá-lo.

Uma outra diferença fundamental entre o software e os demais produtos é o fato de ser particularmente vantajoso para o consumidor final a padronização. A existência de pluralidade de padrões geralmente causa confusão, pois não é viável na prática conhecer todas as tecnologias em profundidade. Concretamente: o mundo seria muito confuso se cada indivíduo ou empresa usassem um computador diferente, com um sistema operacional diferente, com editor de texto diferente, etc. A padronização do software faz facilitar a árdua tarefa de se aprender uma nova e complexa forma de se trabalhar, e mesmo de se pensar. A existência de pessoas próximas com as quais pode-se tirar dúvidas a respeito do uso do software facilita muito a efetiva assimilação do seu uso. Em outras palavras: se um software é muito usado, existe boa probabilidade de se ter um amigo que conhece detalhes de sua utilização.

O software adquire valor adicional pelo fato de ser muito usado.

Na verdade, essa característica do software, em maior ou menor nível, ocorre com produtos do tipo “conteúdo”. Um jornal tem mais valor se tem mais leitores. Um filme ou programa de TV tem mais valor se tiver mais espectadores. Um livro tem mais valor se tiver mais leitores. O autor ou produtor do conteúdo quer ganhar dinheiro com a venda e distribuição do conteúdo. Contudo, estabelecer uma forma demasiado severa de evitar cópias e observação do conteúdo pode eventualmente ser contra os objetivos do autor ou produtor.

Se um consumidor escolher uma televisão da marca x , a probabilidade de que a próxima televisão seja da mesma marca é relativamente pequena. Há várias marcas disponíveis, e muitas delas são muito boas. O produto “televisão” é uma mercadoria. Mas se um consumidor escolhe um software para resolver um problema, se o software é satisfatório, a tendência é o consumidor recomendar sempre o mesmo software para resolver o tal problema. É fácil entender o motivo desse comportamento. A complexidade do uso do software pode ser grande, e portanto há um esforço em se aprender a usá-lo bem. Esse esforço pode ser bastante grande. O consumidor tende a recomendar o software que já usou para não jogar fora o investimento de tempo que já fez em aprender a usar o software.

Outra forma de entender a diferença fundamental entre software e a maioria dos outros produtos é a seguinte. Seria possível um futuro em que produtos como televisão, gasolina, energia, roupas, etc. fossem oferecidos gratuitamente ? É muito difícil imaginar que isso pudesse acontecer. No entanto, é perfeitamente possível que exista software gratuito. Se o criador de um software não ganha dinheiro diretamente distribuindo um software gratuito, que interesse haveria em fazê-lo ? Muito simples: sendo gratuito, o consumidor torna-se mais interessado em experimentar o software. Assim, facilita-se a criação de uma base

de usuários do software, e portanto o produto fica mais confiável e também mais valioso. Com a base de usuários consolidada, é muito possível pensar em formas indiretas de se ganhar dinheiro. Um exemplo disso é o linux. Discussão adicional sobre a conveniência de se doar conteúdo no ambiente atual é feita na seção 2.18.2, na página 58.

2.9 Analogia entre biologia e negócios

O ambiente biológico e o ambiente de negócios possuem inúmeras características comuns. Quem se interessa por um desses assuntos tem muito a aprender por estudar o outro. Um dos pontos onde há muita semelhança entre esses dois assuntos é a forma como ocorre evolução nos dois ambientes.

Para o ambiente biológico, existe a “teoria da evolução” que resumidamente diz o seguinte. As espécies entram em competição por recursos de existência (como comida, água, território). Os menos adaptados não conseguem obter suficientes recursos e tendem a morrer. Suas chances de se reproduzir são pequenas. Os melhores adaptados sobrevivem mais facilmente, e conseguem gerar descendentes. Nessas circunstâncias, ocorre uma “seleção natural”, que reforça as características que se favorecem a sobrevivência. Repetido o processo seguidamente de geração a geração, gradualmente forja-se novas espécies, com reforçadas características que favorecem a vida e a reprodução no ambiente em que vivem. O planeta tem uma história extremamente longa. Os integrantes do ambiente biológico que existem hoje são aqueles que resultaram da evolução desde que surgiu vida no planeta, não se sabe exatamente como, há bilhões de anos.

Seja a seguinte analogia entre biologia e negócios: [uma “espécie” corresponde a uma “empresa”]. Assim como as espécies, as empresas vivem num ambiente em que competem por recursos essenciais. Uma espécie pode se manter viva até por milhares de anos, desde que seus indivíduos consigam se reproduzir. Uma empresa teoricamente pode se manter viva indefinidamente, mesmo precisando renovar seus funcionários e colaboradores. Para isso, é preciso que garanta o seu acesso a recursos essenciais de existência.

As espécies, assim como as empresas, apesar de todas as variações que existem, tem uma missão comum: sobreviver. Para sobreviver, a empresa busca o lucro. Manter o fluxo de caixa o mais azul possível (isto é: fazer bastante dinheiro) é um objetivo de praticamente qualquer empresa. Esse objetivo, no entanto, não se justifica por si só. O objetivo de uma empresa que se justifica por si só é o da sobrevivência. Se o objetivo das empresa fosse somente o lucro financeiro, então não haveria motivo para que uma empresa fazer filantropia. No entanto

ocasionalmente uma empresa faz filantropia, tal como doar equipamentos para uma comunidade pobre¹.

Garantir a sobrevivência sob competição brava, pode eventualmente não ser possível. Nesse caso, o resultado é a extinção da espécie ou quebra da empresa.

2.9.1 Zona de conforto

Seja a definição de *zona de conforto*: A zona de conforto de uma espécie (empresa) é a diferença entre a capacidade que se tem de uma habilidade essencial e a quantidade mínima dessa capacidade para que se possa sobreviver.

Seja o caso leão-zebra. Para um leão, que precisa correr atrás de uma zebra para come-la, a habilidade essencial é correr. A velocidade que se corre é a capacidade que o leão tem dessa habilidade. A zona de conforto é a diferença entre a velocidade do leão e a velocidade da zebra.

Seja o caso de produção de bem de consumo. Para um produtor de monitor de computador (ou quase qualquer outro bem de consumo), que precisa vender a produção para manter-se, a habilidade essencial é o ato de produzir. O preço do produto é a capacidade que o empresário tem dessa habilidade. A zona de conforto é a diferença entre o custo de produção e o valor que o mercado está disposto a pagar.

Uma característica interessante da teoria da evolução é que em geral², observado-se o ambiente em escala de tempo longa o suficiente, **o processo evolutivo tende a fazer a zona de conforto reduzir-se a zero.**

No caso do leão e da zebra, o leão come as zebras que correm pouco, e portanto sobrevivem as que correm mais. Essa condição que seleciona as que correm mais cria uma pressão evolutiva na espécie zebra. Com isso, o processo evolutivo leva a espécie zebra passa a correr cada vez. Observado em escala de tempo longa o suficiente, o processo ocorre até que eventualmente um leão que corre menos que a média não consegue mais pegar zebras, e morre de fome. A zona de conforto do leão foi a zero. Essa mesma pressão evolutiva faz com que o leão seja selecionado para correr cada vez mais. Trata-se de uma situação de “maratona evolutiva”, em que tanto o leão quanto a zebra são selecionados para correrem cada vez mais.

No caso do produtor de monitor de computador, a observação de que há mercado para o produto atrai novos entrantes na atividade de produzir. Isso

¹ Não pretendo entrar em discussão sobre *todos* os motivos pelos quais uma empresa faz filantropia. Apenas gostaria de enfatizar que a motivação essencial de empresas é o da sobrevivência.

² Não “sempre”, porque podem surgir condições que perturbam o processo de evolução.

leva a produção a gradualmente aumentar, e pela lei de oferta e procura o mercado gradualmente passa a pagar menos pelo produto. O processo desenvolve-se até que eventualmente um dos produtores não consiga mais pagar seus custos, e quebre. A zona de conforto do produtor foi a zero. A pressão evolutiva do mercado (em tese) seleciona as empresas que conseguem produzir por custo mais baixo. Essa situação faz uma pressão evolutiva em que as empresas precisam produzir de forma cada vez mais eficiente, isto é: a custos baixos.

O leitor pode estar se perguntando: o que tudo isso tem a ver com C++ ? Resposta: veja a seção 2.24, na página 83.

2.10 Resultado da competição: resta um ou poucos

Quando uma empresa dedica-se a um produto tipo “conhecimento empacotado”, que tem custos de desenvolvimento altos comparado com os custos de produção, cria-se uma situação evolutiva no mercado que em geral, passado tempo suficiente, faz quebrar quase todas as alternativas e a **restar um ou poucos**. Isso ocorre com sistema operacional de computador, por exemplo. Ocorre também com editor de texto, com software de mensagem instantânea, com console de jogos, com definição do hardware básico (arquitetura do computador, CPU, etc.), com software cliente de email, com navegador de Internet, etc.

Uma empresa que tem um produto inovador de software deve ter uma estratégia para fazer o seu produto ser aceito no mercado. Faz sentido procurar explorar o mercado rapidamente, pois do contrário, caso o nicho de mercado seja efetivamente lucrativo, haverá uma atração de concorrentes. Alguns autores chamam essa estratégia de *get big fast* (tornar-se grande rapidamente). Quem conseguir crescer no momento em que o mercado está virgem, obterá vantagem competitiva sustentável. Especialmente se conseguir obter maioria da fatia do mercado, e com isso tornar-se referência natural no segmento.

O mercado se confunde com excesso de opções, e em geral acaba se concentrando em uma ou poucas opções, depois de vencido o período de maturação do segmento de mercado.

2.11 Melhores práticas

Por observar as técnicas de administração de diversas empresas, cruzar esses resultados com o que se define ser “sucesso” e por interpretar esses dados, pode-se determinar um conjunto de práticas administrativas que tem mais possibilidade de conduzir ao sucesso. Se o universo de empresas observada é suficientemente grande, e se os dados (evidências) fazem emergir um padrão significativo de práticas associadas ao sucesso, pode-se dizer que as tais práticas são as “melhores práticas” (*best practices*). Conhecimento desse tipo é de

interesse de um estudioso do ramo de “administração”, digno de publicação de livros ou artigos em revista. Um chefe de empresa é em princípio um consumidor desse tipo de conhecimento.

Por exemplo: suponha que um estudioso relacione o sucesso de empresas de software com o número de níveis de hierarquia da empresa. Se as evidências mostrarem que ter relativamente poucos níveis (menos níveis que uma empresa de outro setor e faturamento semelhante) é associado ao sucesso da empresa, então se pode dizer que “ter poucos níveis de hierarquia numa empresa de software é uma *best practice*”.

A busca por melhores práticas em diversos setores de administração é uma busca interminável dos estudiosos de administração. Desde que o ambiente econômico muda constantemente, segue sendo interessante repetir a análise das melhores práticas. Talvez uma prática tenha sido muito boa até um certo momento, e a partir de um determinado ponto, devido a mudanças no ambiente, a mesma prática já não seja boa.

Por exemplo: Suponha que o estudo de melhores práticas indique que empresas grandes e com atuação global tenham vantagem competitiva sobre empresas menores para implantação de sistemas corporativos. Há mudanças recentes do ambiente, incluindo a consolidação da infra-estrutura da web, nível de maturidade de software gratuito, nível de conhecimento de tecnologias básicas por empresas locais, baixo nível de capitalização das empresas que ainda não estão informatizadas, etc. Pode ser que nesse novo ambiente empresas integradoras de solução de porte pequeno, grande agilidade e suficiente competência técnica sejam mais competitivas que empresas grandes e de atuação global. Para que se avalie qual a situação que produz mais crescimento empresarial (condição definida como “sucesso”), requer-se um estudo atualizado. Esse exemplo na verdade ilustra com pequena distorção o conceito de “melhores práticas”, pois o atributo estudado é o “tamanho da empresa”, e isso não é uma “prática administrativa”. Contudo, o exemplo ilustra bem a necessidade de estudos atualizados sobre a condição do ambiente econômico.

Outro ponto de interesse para desenvolvedores de software é estudar as melhores práticas do ato de desenvolver software em si. Há práticas controversas como “extreme programming”. Essa prática deve ser considerada como uma *best practice* para a atividade de desenvolvimento de software ? O que dizer do uso de software de gerenciamento de código fonte (e.g. Source Safe da Microsoft) ? O que dizer da atividade de rever código fonte ?

2.12 Incorporação de novidades

Há estudos que mostram como uma novidade costuma ser incorporada por um grupo social. Pode-se classificar (de forma simplificada) os usuários em 4 grupos, com suas porcentagens:

1. A - Desbravadores (10%)
2. B - Primeira maioria (40%)
3. C - Maioria atrasada (40%)
4. D - Retardatários (10%)

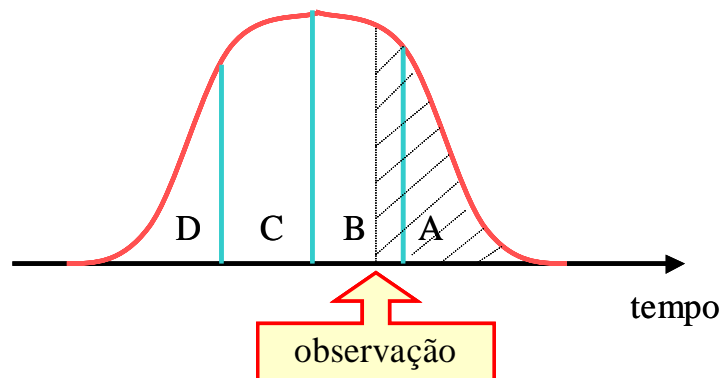


Figura 2: Diagrama mostrando a porcentagem de usuários que incorporam uma novidade em relação ao tempo.

Em software, os desbravadores são os que estão sempre testando software beta, e correndo atrás de todo tipo de novidades. Os da primeira maioria são aqueles que lêem os comentários feitos pelos desbravadores e identificam as tendências consistentes e minimamente maduras, e incorporam-nas. Os da maioria atrasada são os que “correm atrás” dos outros, e apressam-se a incorporar as novidades quando já há muita gente usando-as. Os retardatários são os últimos a adotarem a novidade, e apenas o fazem quando é absolutamente evidente que precisam fazê-lo, sendo que quase todos já o fizeram. As porcentagens de pessoas em cada uma das categorias é mostrado acima.

É interessante cruzar a informação de classificação de usuários com o perfil de sucessos. A experiência mostra que a maior parte dos indivíduos e firmas bem sucedidos são os que se comportam como “primeira maioria”, e não como “desbravadores”. Enquanto os “desbravadores” freqüentemente gastam tempo precioso para depurar problemas de novidades, os da primeira maioria optam pela novidade quando já há cultura suficiente para utilizar a novidade de forma efetiva.

Há um exemplo antológico no Brasil que ilustra a eventual falta de sucesso de quem se comporta como desbravador. Em meados dos anos 1980, dizia-se que os computadores de grande porte (*main frame*) estavam fadados à extinção. O destino seria substituí-los por versáteis estações de trabalho (*workstation*). Os bancos tradicionalmente são grandes consumidores de computadores grande porte, e de tecnologia de informação em geral. O Banco Bamerindus decidiu investir uma grande soma num projeto “desbravador” que iria substituir a operação principal do banco, que era feita por computadores de grande porte,

para passar a ser feita em estações de trabalho. Para resumir a história: o projeto não funcionou adequadamente, e a experiência confirmou a suspeita de vários gerentes de tecnologia, de que os bancos operam melhor com computadores de grande porte, e não com estação de trabalho. O Banco Bamerindus sofreu um grande prejuízo com o projeto, e acabou sendo vendido pouco tempo depois para o atual HSBC. O gerente de tecnologia de banco que viveu essa época, e acertou, foi o que disse: “vou esperar um pouco antes de abandonar o computador de grande porte e adotar a estação de trabalho” (comportamento de primeira maioria), e não o que disse “vou partir imediatamente para o projeto de abandonar o computador de grande porte e adotar estações de trabalho” (comportamento de desbravador).

A dinâmica de incorporação de novidades, portanto, começa pelos desbravadores, seguindo-se da primeira maioria, maioria atrasada e retardatários. Uma firma ou indivíduo que queira tornar uma alternativa tecnológica efetivamente usada no mercado deve ser capaz de influenciar os desbravadores, para que então a dinâmica de incorporação de novidades siga seu caminho natural. Em outras palavras: pode-se dizer que há dois tipos de usuários:

- Usuário influenciador
- Usuário comum.

O usuário influenciador é tipicamente um desbravador que é influente, tal como um jornalista ou consultor famoso. Trata-se de um usuário que observa atentamente as reportagens dos desbravadores e sintetiza essa informação de forma interpretada para produzir conclusões que são por sua vez observadas pela primeira maioria. As revistas especializadas trazem sempre reportagens sobre novidades tecnológicas.

Há uma categoria de profissionais que precisa tomar decisões sobre uma tecnologia a adotar para solucionar determinado problema. São por exemplo os gerentes de tecnologia de empresas. Ter capacidade de convencer esse tipo de pessoa é absolutamente importante para o sucesso comercial de uma firma de software, pois são eles quem decidem como alocar recursos financeiros da firma para adquirir tecnologia no mercado.

O termo “novidade” não é sinônimo de “qualidade”, especialmente no mercado de tecnologia de informação. Em muitas empresas ou entidades em que desenvolvem sistemas com alto requisito confiabilidade, por exemplo a NASA, é norma evitar-se o uso de tecnologias lançadas recentemente no mercado. Há um tempo mínimo de existência de uma tecnologia para que se possa considerá-la plenamente confiável. Esse tempo são alguns anos. Um profissional ou empresa com atuação em TI deve manter-se atento as novidades. Mas estar atento *não* significa adotar imediatamente tudo o que o mercado inventa, e viver num

frenesi de upgrades. Significa ler muito sobre o mercado, especialmente os comentários positivos e negativos sobre as novas tecnologias.

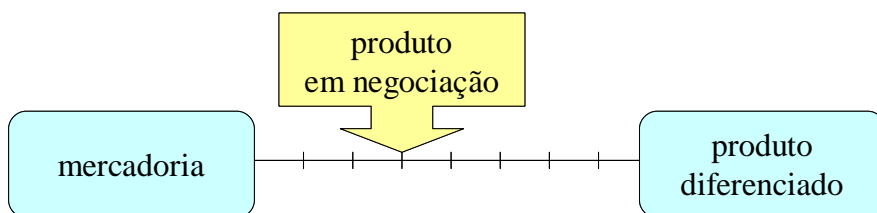
*Há algo melhor que aprender com os próprios erros.
É aprender com os erros dos outros.*

Em 1995, pouco antes de a Microsoft lançar o Windows95, havia um debate acalorado no mercado se deveriam usar o Windows 3.1 (última versão disponível então), ou se deveriam usar o OS/2 da IBM. O OS/2 era claramente superior ao Windows 3.1. Mas o mercado sabia que a Microsoft iria lançar em pouco tempo o Windows95, e que as duas plataformas seriam incompatíveis. A evolução do mercado provavelmente faria restar apenas um dos dois (veja seção 2.10). De fato, passado tempo suficiente, o OS/2 praticamente desapareceu. Quem na época foi desbravador e estudou o OS/2, acabou fazendo mal uso do tempo, pois o legado de usar OS/2 é pouco útil nos dias de hoje.

Quem usa C++, em geral aproveita bem o legado de uma atividade para a seguinte.

2.13 Mercadoria e produto diferenciado

Quando ocorre uma transação econômica, há dois atores – o comprador e o vendedor. O produto que está sendo negociado está em algum ponto de uma escala, cujos extremos estão marcados com: “mercadoria”³ e “produto diferenciado”.



Quando o produto é uma mercadoria, quem está em boa situação para o negócio é o comprador. Quando trata-se de um produto diferenciado, a situação se inverte.

A capacidade profissional de um indivíduo é um produto no mercado. O preço é a remuneração. Para o profissional pretender obter boa remuneração, é necessário que esse profissional seja um “produto diferenciado no mercado”.

³ “mercadoria” em inglês é *commodity*

2.13.1 Mercadoria escassa

Quando uma mercadoria é necessária e escassa, o seu valor tende a subir. Quem pode suprir a mercadoria escassa pode praticar um preço que lhe é vantajoso (se for bom negociador). Mas vivemos numa sociedade com abundância de informação. Se uma mercadoria de boa demanda no mercado torna-se escassa e seu preço sobe, muitos saberão disso. Portanto, será provavelmente bastante lucrativo investir no sentido de atender a demanda. Como o mercado financeiro está bastante desenvolvido, pode-se financiar o suprimento da demanda rapidamente. Adicionalmente, o setor de transportes está também bastante desenvolvido, de forma que é cada vez mais viável produzir numa parte do mundo e levar o produto até um local bastante distante. Costuma haver bastante empreendedores dispostos a quase todo tipo de investimento. Portanto, com o tempo a tendência é que uma mercadoria torne-se plenamente atendida, e portanto o preço de comercialização tende a cair até o mínimo possível para o mercado em questão.

Seja o estudo de caso a seguir. Desde que a Internet tornou-se muito importante para empresas, em torno do ano de 1995, surgiu demanda para desenvolvimento de páginas e sistemas de web. Surgiu demanda para desenvolvimento de páginas html. Como essa profissão não existia até então, os poucos que sabiam fazer o serviço tornaram-se mercadoria escassa, e puderam cobrar alto. Há não muito tempo atrás, o simples desenvolvimento de uma página html simples era objeto de excelente remuneração. Com essa situação bastante reportada pela imprensa, surgiu interesse em criação de cursos para webdesign, e também desenvolvimento de software para facilitar o desenvolvimento de html. Tudo isso foi feito e hoje há muito mais profissionais habilitados a desenvolver páginas html, usando software muito mais fácil de usar. Com isso, o preço do profissional de webdesign caiu bastante, mesmo com uma demanda sólida por profissionais desse tipo.

Uma das características na nossa época é que quase tudo tornou-se uma mercadoria, inclusive nossa capacidade profissional. Essa é uma das consequências do fenômeno econômico conhecido como “globalização”. Como estamos comentando, a longo prazo, a única forma de se manter um preço *premium* para um produto é torna-lo um produto diferenciado.

Portanto, qual deve ser a estratégia de um profissional de tecnologia de informação, ou de uma empresa, para evitar ser transformado numa mercadoria e portanto jogado numa situação onde ganhará pouco ? É evidentemente uma pergunta complexa, mas a resposta certamente estará na capacidade de o profissional ou empresa reciclar-se mais rápido que a média, mantendo-se sempre como um produto diferenciado. Na hora de escolher uma linguagem de programação para estudar, procure pensar a longo prazo e avaliar em quais mercados pretende atuar.

2.13.2 Estratégia lucrativa

Uma “estratégia lucrativa”, ou para ser mais explícito: uma “receita para ficar rico”, é algo que basicamente muitos gostariam de conhecer e adotar. Muitos autores dedicam-se a escrever sobre esse assunto. Quando se lê num livro algo do gênero, a essência do que se diz geralmente enquadra-se em uma das categorias abaixo.

- A estratégia é uma obviedade, por exemplo: “satisfaça seu cliente”. É difícil extrair dela algo de realmente útil. Se o autor realmente tivesse uma estratégia exequível, provavelmente não a publicaria. Portanto uma divertida interpretação sobre o que seria a verdadeira receita para ficar rico é: “escreva um livro sobre como ficar rico, e com isso o autor fica rico por transferir dinheiro dos leitores para si”.
- A estratégia é fácil de se falar mas difícil de se fazer. Contudo é importante tê-la em mente no momento de tomada de decisão.

Heis uma estratégia lucrativa particularmente interessante, correspondente ao último dos casos acima.

Compre mercadorias e venda produtos diferenciados

Como já se mostrou, a posição econômica de vantagem no caso de compra ocorre para produtos tipo “mercadoria”, e para venda ocorre para “produtos diferenciados”. Portanto, adotando essa estratégia se está em vantagem econômica em ambos os lados.

Todos nós compramos e vendemos regularmente na sociedade em que vivemos. Muitos acham que não vendem nada, mas se esquecem que a capacidade de trabalho é um serviço que se vende no mercado de trabalho. Para se ter sucesso profissional num ambiente competitivo, é preciso que essa venda seja de um produto diferenciado. Ou seja, trabalhar numa profissão em que há pessoas demais (possivelmente devido a uma barreira de entrada pequena para novos profissionais dessa categoria) não é receita para ganhar bom dinheiro como assalariado. Em geral um assalariado não ganha um dinheiro realmente bom, mas há vários casos em que ganha-se satisfatoriamente, e mesmo alguns casos em que se ganha bastante bem. Num ambiente competitivo, quanto mais diferenciado for o profissional, tanto mais provável será obter boa remuneração (desde que haja demanda da especialidade em questão).

Um problema para adotar a estratégia proposta é a necessidade de que algum outro ator do cenário incorpore a situação inversa, isto é, comprar produtos diferenciados e vender mercadorias. E quem não nasceu ontem já aprendeu essa lição há muito tempo. Ou seja, em geral os demais atores do cenário econômico

com quem nós nos relacionamos tendem a adotar essa estratégia a favor deles, e portanto adotar a estratégia no sentido oposto aplicado a nós.

O Brasil vende aviões Embraer, e compra cevada (que não cresce bem no Brasil devido ao clima) para produzir cerveja. Esse tipo de relação comercial é plenamente de acordo com a estratégia lucrativa proposta. O problema é que a proporção com que se adota essa estratégia no sentido certo é pequena. Com exceção de exemplos honrosos (como citado), boa parte da participação Brasileira no mercado internacional se dá por exportar produtos sem marca (como soja, frango, suco de laranja), e importar insumos e produtos tecnológicos (inclui pagamento de royalties). Conclusão: outros países adotam a estratégia lucrativa corretamente, colocando o Brasil na situação inversa.

Voltemos à computação. Reconheça-se: um computador transformou-se numa mercadoria cada vez mais barata. Com exceção de alguns detalhes de hardware (CPU ou disco rígido particularmente rápido, placa de vídeo ou de som com alguma característica especial, etc.), um computador é basicamente idêntico a outro, uma autêntica mercadoria. E está ficando cada vez mais difícil convencer um comprador de computador a pagar preço *premium* por um desses detalhes de hardware.

Portanto, como seria a aplicação da estratégia em questão para o mercado de tecnologia de informação ? Seja o exemplo: compra-se computadores, instala-se neles software que se desenvolveu, e vende-se um sistema para uma empresa. Considerando que o software em questão possui grande valor, a empresa que vende a licença do software é quem está vendendo o produto diferenciado e bem remunerado. O computador é “qualquer PC com a configuração padrão” (claramente uma mercadoria). Portanto, **desenvolver software e vender soluções completas** utilizando o software que se desenvolveu, ainda que requeira-se a aquisição de hardware, é plenamente compatível com a estratégia lucrativa proposta. Em contraste, revender software de alto valor desenvolvido por outro não é em princípio uma atividade de alta agregação de valor. Mas eventualmente poderia se-lo, se a atividade de configuração do software, de análise de sistema, ou a consultoria que se vende junto com o software for de alto valor.

2.14 O objetivo é atender ao cliente ?

Seja o estudo de caso hipotético. Um cliente deseja um sistema especificado de TI, chamado de *xyz*, de alto valor. Pode-se fazer o sistema “na mão”, usando ferramentas de produtor de tecnologia, ou pode-se fazê-lo por comprar uma excelente e cara ferramenta de uma grande empresa.

No primeiro caso, será necessário estudar o problema a fundo e agregar diversos módulos, desenvolvendo vários deles. Nessa abordagem, a remuneração irá para os analistas e programadores, e como resultado, além de se atender ao cliente,

haverá um aumento de conhecimento e experiência dos profissionais envolvidos. O preço das ferramentas de produtor de tecnologia são relativamente baratas, se não gratuitas.

No segundo caso, é preciso comprar a ferramenta cara e excelente da grande empresa. Obtém-se a solução final por usar os padrões pré-programados da tal ferramenta, acrescentando-se alguns ajustes para que fique exatamente como o cliente pediu. Nessa abordagem, o nível de conhecimento exigido para se configurar a solução final usando-se essa ferramenta é bem menor que no primeiro caso. Portanto, paga-se pela ferramenta, mas em compensação economiza-se no preço dos profissionais envolvidos. Caso exista algum bug, ou necessidade de suporte, recorre-se ao produtor de tecnologia, que se for bom irá atender a solução e aprender com ela, para melhorar o produto em novas versões.

Observando as duas abordagens a longo prazo, conclui-se que no primeiro caso cultiva-se conhecimento de produtor de tecnologia. No segundo caso apenas aprende-se a usar a ferramenta feita por empresa externa. Com o tempo, a tendência é o preço da ferramenta sofisticada aumentar, e a diminuir a remuneração dos profissionais que a operam (pois vai tornando-se cada vez mais fácil um novo entrante atuar como operador da ferramenta, e a concorrência faz a remuneração diminuir). Em ambos os casos o cliente final é atendido, mas no primeiro caso a empresa integradora de solução garante uma parcela importante da remuneração a longo prazo, enquanto no segundo caso torna-se difícil negociar boas remunerações pela tarefa de operar a ferramenta da grande empresa.

Reflita-se novamente: o objetivo da empresa apenas o de atender ao cliente ?

*O objetivo pode não ser tão somente atender ao cliente, mas
faz-lo
de forma a consolidar-se como fornecedor de elemento
de alta agregação de conhecimento, e portanto de valor.*

Quando um nicho de mercado se abre, em pouco tempo as grandes empresas - que tem amplos recursos de desenvolvimento - lançam produtos para atender a demanda. Um caso típico é o uso de XML. Há produtos de grandes empresas que automatizam de forma elegante e fácil de operar a produção de soluções para comércio eletrônico baseado em XML. Os produtos são fabulosos, mas seu preço é caro. Quando um integrador de solução propões um sistema baseado nesses produtos, há que se incluir no preço a compra de licença da ferramenta.

Para quem geralmente usa C++ e cultiva cultura de produtor de tecnologia, pode-se pensar em gerar soluções equivalentes baseado em bibliotecas gratuitas que processam XML. Dessa forma, atende-se o cliente num contexto em que a participação do integrador de solução é altamente agregadora de conhecimento, e portanto de valor. O cliente tem sua solução, e quem ganhou dinheiro foi o integrador de solução, e não a grande empresa que produz a ferramenta. Além disso, quanto mais se fizer esse tipo de serviço, tanto mais conhecimento se adquire em fazê-lo. Aos poucos, pode-se criar ferramentas de uso da empresa, que semi-automatizam a produção de solução. Eventualmente, pode-se até comercializar essa ferramenta, e nesse caso se estará concorrendo diretamente no mercado de produção de tecnologia.

2.15 Vantagem competitiva

A vantagem competitiva é a condição que coloca o fornecedor de um produto em boa posição para vencer os concorrentes. Por exemplo: possuir um clima adequado a produção de um determinado vegetal é uma vantagem competitiva. Quem não possui o tal clima teria que gastar dinheiro gerar artificialmente o clima adequado, ou produzir com menor produtividade. Outra vantagem competitiva pode ser o fato de haver afinidade cultural para um produto penetrar em um mercado.

O que as empresas e indivíduos buscam idealmente são vantagens competitivas que sejam sustentáveis. No mundo atual, com abundância de informações, mercado financeiro desenvolvido e sistema de transportes e comunicações eficiente, muitas barreiras de entrada para novos competidores praticamente desapareceram. Nessa situação, muito do que tradicionalmente era considerado como vantagem competitiva sustentável não funciona mais.

Dentre as coisas que ainda funcionam como vantagem competitiva sustentável, uma delas é a marca. O consumidor que tem referências mentais positivas de uma certa marca, tende a escolhê-la a despeito da existência de marcas concorrentes, e não raro o fazem ainda que exista diferencial no preço. Se o produto em questão é do tipo conhecimento empacotado (veja página 33), o comprador tem motivos adicionais para não mudar de marca. Nesses casos, em geral, o cliente investe tempo em aprender a usar o produto, e há uma tendência em não mudar de marca para preservar o seu investimento em usar aquele produto diferenciado.

Ao montar uma estratégia para si, ou para a empresa que se representa, procure pensar em quais vantagens competitivas se pretende cultivar. No mercado tumultuado de tecnologia de informação, a experiência mostra que num ciclo de vida de um profissional, o ambiente muda várias vezes. A capacidade de adaptação é chave para quem quer manter-se competitivo no mercado. Muitas linguagens de computador e outras tecnologias nasceram e morreram. O profissional precisa decidir onde investir seu tempo de estudo.

Surge a pergunta: o que é relativamente perene e estável no meio de tantas mudanças ? A linguagem C/C++ foi inventada desde a estaca zero para que seja relativamente fácil se fazer um compilador para ela. Além disso, a respeitável entidade GNU há muito tempo distribui um compilador C/C++ de excelente qualidade, com o fonte incluído, e versão para várias plataformas. As novidades continuam a surgir, mas é quase certo que para qualquer computador ou processador novo que alguém invente, haverá rapidamente um compilador C/C++ para ele. Dentre as novidades relativamente recentes que demandam desenvolvimento de software, e para as quais há compilador C++ disponível, cita-se:

- Palm Pilot.
- Pocket PC (Windows CE).
- Chips microcontroladores e DSP, úteis para eletrônica embarcada, processamento de sinais, controle e eletrônica embarcada.
- Plataforma .Net (além das linguagens C#, J# e Visual Basic, pode-se também programar para essa plataforma com a boa e velha linguagem C++).
- Tecnologia de voz (reconhecimento e síntese de voz).
- Redes neurais.
- XML e *business to business*.

O conhecimento e cultura no uso de C++ é algo que somente se obtém com tempo. Na ocasião do surgimento de um novo segmento no mercado de TI, em geral inicialmente existe a disponibilidade apenas da linguagem C/C++. Se o segmento se mostrar interessante (isto é, rentável), isso significa oportunidade para que empresas lancem ferramentas e linguagens que facilitem o desenvolvimento de software para esse novo segmento. Dessa forma, o desenvolvimento dessas ferramentas torna-se ainda um outro segmento a cobrir. O desenvolvimento de ferramentas é em geral feito em C++.

Em resumo, conhecer C++ é uma vantagem competitiva para que um profissional ou empresa se posicione para cobrir rapidamente um novo segmento que esteja em fase de consolidação. E essa é a fase ideal para se

2.16 Sociedade do conhecimento

A humanidade está vivendo atualmente uma condição social que muitos autores chamam de “sociedade do conhecimento”. O conhecimento tornou-se o mais valorizado ativo das empresas. Obter, assimilar, classificar, recuperar, acessar, interpretar, gerenciar e manipular conhecimentos são algumas das habilidades valorizadas nesse ambiente. Dentre os conhecimentos que se deve ter não incluem-se apenas os conhecimentos técnicos. Igualmente importante é ter

conhecimentos sociais (relacionamentos), conhecimentos de mercado, conhecimento de como e onde se produz e se acessa (ou se compra) conhecimentos de boa qualidade, saber distinguir um conhecimento de boa qualidade de um conhecimento de qualidade desconfiável, saber para que servem os conhecimentos técnicos que se dispões, etc.

Alguns conceitos fundamentais:

- Dados: é um valor derivado da observação de objeto ou referência. Exemplo: 15 graus celcius. Para serem registrados, guardados e disponibilizados os dados devem estar ligados a um *quadro referencial*. No caso o quadro referencial é a escala de temperatura.
- Informação: é uma coleção de dados **relacionados** que, dentro de um determinado **contexto**, podem transmitir algum significado a que os consulta. Exemplo: guia telefônico, tabela de temperaturas. Para quem não sabe o que é um telefone, o guia telefônico não contém informação alguma.
- Comunicação: é a transferência de informações de um agente de processamento dessa informação para outro.
- Conhecimento: É a familiaridade, a consciência ou entendimento conseguido seja através da experiência ou do estudo. O conhecimento pode ser classificado nas formas abaixo:
 - Tácito - é algo pessoal, formado dentro de um contexto social e individual. Não é propriedade de uma organização ou de uma coletividade. Em geral é difícil de ser transmitido. Por exemplo: um craque de futebol sabe dar uma bicicleta e marcar um gol. Por mais que se veja e reveja o craque fazendo isso, é difícil fazer o mesmo.
 - Explícito - envolve conhecimento dos fatos. É adquirido principalmente pela informação, quase sempre pela educação formal. Está documentado em livros, manuais, bases de dados, etc. Por exemplo: o conhecimento da sintaxe e da programação de C/C++ é um conhecimento explícito. Já a boa programação é em parte explícito, mas envolve uma certa dose de conhecimento tácito. A capacidade de trocar informações com um grupo (tribo) do mesmo segmento de tecnologia é um complemento fundamental para o profissional, e envolve muito conhecimento tácito.
- Sabedoria: É a qualidade de possuir grande conhecimento, geralmente de mais de uma área do saber. A associação de compreensão de diversos conhecimentos faz surgir uma “compreensão geral”, ou “compreensão da grande figura”, que é exatamente o aspecto que configura a sabedoria. Uma das vertentes da sabedoria é conseguir ver o mundo pelo ponto de vista de outras pessoas.

2.16.1 Mudança do paradigma de negócios

Pode-se chamar de “sociedade do conhecimento” a sociedade que em grande parte adotou para si os “paradigmas economia do conhecimento” (mais atuais), que se opõem aos “paradigmas da economia do industrial”. Um quadro comparativo desses paradigmas é mostrado abaixo.

Paradigmas da Economia Industrial	Paradigmas da Economia do Conhecimento
Fordismo, economia de escala, padronização	Escala flexível, customização, pequenos estoques, <i>just in time</i> .
Grandes tempos de resposta	Tempo real
Valorização de ativos tangíveis	Valorização de ativos intangíveis
Atuação local, regional	Atuação global
Intensivo em energia (petróleo) e materiais	Intensivo em informação e conhecimento
Ramos matrizes: petrolíferas, petroquímicas, automóveis, aeroespacial	Ramos matrizes: eletrônico, informática, telecomunicações, robótica, genética
Qualidade do produto	Qualidade do processo (além de qualidade do produto)
Grandes organizações verticais	Sub-contratação (tercerização), <i>downsizing</i> , redes de relacionamento entre empresas
Rigidez da hierarquia administrativa, separada da produção	Flexibilidade e integração de empresa com clientes, colaboradores e fornecedores
Taylorismo. Decomposição do processo de trabalho	Trabalhador polivalente
Empregado como gerador de custo	Empregado como geradores de receita (capital intelectual)
Pleno emprego	Empregabilidade
Pouco ou nenhum aprendizado no trabalho	Aprendizado contínuo
Qualificação formal	Competência (capacidade efetiva de resolver

2.16.2 Mercadorização

Tradicionalmente, se um segmento do mercado estiver na situação de escassez e com ganhos relativamente altos, esse segmento torna-se atrativo para negociantes. Desde que exista informação disponível sobre o ambiente de negócios, a tendência é de o segmento atrair novos entrantes, que farão aumentar a oferta de produtos, até que o preço se estabilize num patamar razoável (com taxas de lucro relativamente pequenas), quando então diz-se que o segmento de mercado está maduro.

No ambiente atual, a Internet ajudou a tornar as informações mais abundantes e baratas. Portanto como nunca há reportagens informando qualquer segmento de mercado que esteja “dando dinheiro”. Além disso, a facilidade de comunicação tornou o conhecimento codificável rapidamente transferível de um local a outro. Com isso, serviços de conhecimento passaram a ser globalizados. Por exemplo: pode-se desenvolver software num país para um cliente em outro país.

O efeito final desses fatores é a chamada “mercadorização” (*commoditization*) do ambiente de negócios. Isto é, quase tudo transforma-se numa mercadoria, e rapidamente. A concorrência em todos os setores parece ser incrivelmente grande, e o consumidor exige preço e qualidade como nunca, sabendo que há inúmeras opções caso não se sinta satisfeito. Isso é bom para quem compra, mas torna-se difícil a vida de quem quer vender. E todos nós temos que vender alguma coisa, nem que seja a nossa capacidade profissional no mercado.

Num exemplo extremo (mas real) de mercadorização, é um anúncio de “transplante de fígado”, que um hospital espanhol veiculou numa revista de circulação internacional. O exemplo é extremo e merece citação porque o produto em questão requer alta especialização, e é de aplicação um tanto pessoal, por tratar-se de saúde. Veiculado numa revista de circulação internacional, obviamente se está objetivando que um paciente estrangeiro venha ao país com finalidade de sofrer a intervenção.

Estudando o assunto, percebe-se que o transplante de fígado é uma especialidade dentro do segmento de transplantes, já muito especializado. É um procedimento caro e relativamente raro, ainda feito de forma um tanto artesanal. Considerando um fluxo de pacientes um pouco maior, o hospital em questão pode otimizar certas etapas do procedimento e valer-se de economia de escala. O conhecimento para executar o procedimento de forma otimizada é preservado pela equipe, e o paciente apenas recebe o efeito do procedimento. Trata-se de mais um exemplo de conhecimento empacotado.

2.16.3 Remuneração × agregação de valor na cadeia produtiva

O processo de produção de um produto (ou serviço) é geralmente composto de várias etapas, ou elos. Pode-se dizer que há uma “cadeia produtiva” que faz surgir o produto. Com a comercialização do produto final, faz-se caixa (ganha-se dinheiro), com o qual se remunera direta ou indiretamente toda a cadeia produtiva.

Uma das características da sociedade do conhecimento é que as melhores remunerações da cadeia de valor acabam sendo dos elos que agregam alto valor de conhecimento. Por exemplo: na produção de um carro, o ato de montar as peças não agrega tanto valor de conhecimento quando o projeto de um carro novo. E a remuneração do projetista é maior do que a do montador. Na escala mercadoria-produto_diferenciado, o projetista de carro está mais para produto diferenciado, enquanto o montador está mais para mercadoria. Gradativamente, até mesmo o projetista pode tornar-se também uma mercadoria, e passar a ter menores remunerações. Apenas um projetista que saiba fazer algo difícil (entenda-se: que poucos sabem fazer) e útil (entenda-se: valorizado no mercado) consegue obter boa remuneração.

2.17 Políticas de apoio a “economia do software”

Atuar de forma sólida e competitiva no mercado de software é uma chance para se obter prosperidade econômica e bem estar social. Cultivar as condições para atingir esse objetivo é uma estratégia. Os diversos setores da sociedade podem colaborar ativamente com essa estratégia. Nessa seção, discute-se como cada setor pode colaborar.

2.17.1 O que o governo pode fazer

- **Definir como muito pequeno ou zero o imposto para computadores, suas peças, periféricos, etc.** Essa medida é importante porque o computador não deve ser considerado um “bem de consumo”, mas um “fator de melhoria da sociedade”. Em outras palavras, “quanto mais computadores existirem, melhor a sociedade ficará”, pois será possível obter ganhos de eficiência estrutural na sociedade em outros níveis. Por exemplo: desde que existam computadores em quantidade suficiente, pode-se definir um sistema em que a declaração de imposto de renda é feita eletronicamente, o que é extremamente mais eficiente que fazê-lo com tecnologia em papel. Outra expectativa positiva sobre uma sociedade com muitos computadores é a existência de empresas e empregos relacionados a software, aproveitando-se portanto das receitas que decorrem dessa atividade. Como se sabe, os empregos em desenvolvimento de software são considerados como “empregos de boa qualidade”, pois fixam profissionais de bom gabarito intelectual, que atua numa atividade não poluente e relativamente bem remunerada. Os retornos das empresas em atividades

de software potencialmente podem ser muito grandes. Não é coincidência que muitas fortunas recentes foram construídas a partir do desenvolvimento e comercialização de software.

- **Apoio institucional a software gratuito, e/ou software desenvolvido por empresas nacionais.** O governo tem poder normativo. Por normatizar o uso de software gratuito ou de software desenvolvido por empresas nacionais, se está adotando uma política de fomento à economia do software, e adotando uma estratégia que minimiza o pagamento de royalties e/ou licenças para empresas estrangeiras.
- **Apoio diplomático a iniciativas de exportação de software.** Diplomatas empenham-se por garantir mercado para produtos diversos (suco de laranja, aço, calçados, aviões, etc.). Da mesma forma, deve haver pressão política a nível diplomático no sentido de apoiar a atividade de exportação de software, com pagamento de royalties e/ou licenças de software do estrangeiro para empresas nacionais.
- **Política de fomento tecnológico.** Órgãos de apoio e fomento a pesquisa científica e tecnológica podem contemplar explicitamente o fomento de empresas de software. A definição de áreas para concessão de bolsas de estudo, incluindo mestrado e doutorado, é uma forma concreta de definir essa política de fomento. Aprovação de projetos tecnológicos e de parceria com empresas a partir de universidades e centros de pesquisa é outra forma.
- **Política de financiamento.** Ter uma política de financiamento para empresas de software é mais complexo que a mera alocação de recursos financeiros. A atividade de desenvolvimento de software intrinsecamente lida com valores intangíveis. O nível de conhecimento de uma tecnologia é um fator essencial de definição de valor. A definição de garantias para empréstimos no caso de empresas de software deve em princípio ser capaz de aceitar, e portanto de medir, o valor intangível que é o conhecimento de uma equipe. Concretamente: se uma empresa toma um empréstimo para comprar um tear, se a empresa quebrar o banco pode tomar o tear de volta, leiloa-lo e recuperar o dinheiro. Portanto o tear pode ser usado como garantia do empréstimo. Se uma empresa de software toma um empréstimo para pagar profissionais para estudar e desenvolver um sistema, não há como se recuperar diretamente o dinheiro investido. O conhecimento que se obtém do estudo é um ativo intangível. O órgão financiador deve ser capaz de reconhecer e medir os valores intangíveis com os quais se lida na atividade de desenvolvimento de software, de forma a avaliar com sabedoria a garantia para os empréstimos concedidos.

2.17.2 O que os indivíduos podem fazer

- **Manifestar-se pedindo que sistemas e soluções sejam oferecidas em respeito aos padrões abertos.** A tendência natural de empresas produtoras de tecnologia é criar produtos que “fidelizem” o cliente. Na prática, o que ocorre é que surgem sistemas que para serem executados requerem a compra dos produtos das empresas. Essa ação das empresas é perfeitamente legal. E manifestar-se para que não se adote coletivamente esse tipo de solução também é perfeitamente legal. Liberdade e democracia dão trabalho . . .

Em princípio, sistemas na web devem ser desenvolvidos de forma que o usuário final possa usar qualquer navegador, e qualquer sistema operacional. Qualquer característica diferente dessa significa descaracterizar os padrões abertos.

- **Considerar o uso de software gratuito e software baseado em padrões abertos.** Nem sempre é fácil usar software gratuito, pois em geral o software pago é mais fácil de se usar, e já se tem conhecimento do seu uso. Mas há que se considerar que cada um de nós é parte do problema e da solução. O mercado somos todos nós. Se não usamos uma software, ele é abandonado. Se o usamos, ele adquire valor. Se queremos padrões abertos, devemos estar dispostos a usar as soluções que as implementam.

Há páginas e sistemas na web que somente podem ser vistas / usadas por um navegador específico. Uma atitude a fazer nesses casos é ignorar a tal página, e usar um serviço do concorrente.

- **Considerar os produtos de empresas nacionais de software,** caso seja de qualidade e bom preço. Se queremos empregos na área de software, devemos estar dispostos a contratar as empresas locais.

2.17.3 O que as universidades podem fazer

- Seguir cuidando da missão tradicional da universidade:
 - Cultivar excelência técnica e científica.
 - Formar profissionais capacitados, em boa quantidade e qualidade.
 - Produzir, acumular e classificar conhecimentos, disponibilizando-os aos interessados.
 - Obter amplo reconhecimento (certificação internacional, etc.), com vistas a facilitar parcerias com empresas de atuação global.
- Preparar-se para **empreender projetos de software em parceria com o setor privado.** Essa atitude parece fácil, mas nem sempre o é. Os objetivos originais de empresas lucrativas e centros de pesquisa não são os mesmos. Um busca resultados com rapidez - inclusive resultados financeiros. O

outro busca produzir, acumular e interpretar conhecimentos. A tradição brasileira manteve relativamente separadas as empresas das universidades. Para que agora esses dois segmentos passem a atuar em parceria, é preciso aprendizagem e mudança de atitude de parte a parte. A universidade precisa aprender a trabalhar em projetos com prazos apertados e onde não é aceitável chegar-se a conclusões cientificamente interessantes que não produzem resultados.

- Em parte, a finalidade de se fazer projetos com empresas privadas é obter recursos financeiros para investimento na estrutura da própria universidade.
- Manter-se atenta a mudanças no ambiente, e se for o caso alterar o currículo de seus cursos. As eventuais mudanças de currículo devem ser feitas com cuidado e sabedoria. Em princípio deve-se evitar adotar “soluções da moda”. A missão da universidade é dar formação estrutural ao futuro profissional, com ênfase no conhecimentos duradouros, mas sem desatenção a ferramentas do mercado. O estudo de empreendedorismo a nível de ementa curricular pode ser uma contribuição concreta.

2.17.4 O que as empresas podem fazer

- Por seus próprios interesses, empresas integradoras de solução **adotar estratégia de “espírito de produtor de tecnologia”**. Veja seção 2.22.
- **Consumidores de TI podem dar preferência a soluções derivadas de software gratuito e/ou empresas nacionais de software.** Dessa forma, se está investindo no aumento da inteligência local (nacional), ao invés de transferir conhecimento para empresas no exterior. Adicionalmente se está capitalizando financeiramente as empresas do setor, e em última análise toda a sociedade. Com isso, há tendência de melhoria no ambiente econômico em que se atua.
- **Integradores de TI devem considerar o uso de software gratuito.** Como é exposto, trata-se de questão de sobrevivência a longo prazo.
- **Implantar parceria com universidades e centros de pesquisa.** Para que a universidade e a empresa privada passem a atuar em parceria, é preciso aprendizagem e mudança de atitude de parte a parte. A empresa privada precisa aprender a pensar a longo prazo, e investir na produção de conhecimentos cuja aplicação requer um tempo de maturação mais longo que alguns meses. Somente se obtém diferenciação por investir em genuína produção de conhecimento, e não apenas copiar o conhecimento dos outros.
- **Cobrar do governo uma política de fomento a “economia de software”,** nos moldes em que foi exposto em 2.17.1.

2.18 Livro no Século 21

Porque os livros tem ser como sempre foram ?

Será que o ambiente contemporâneo permite um novo modelo de negócio para livros ?

Dentre os negócios que merecem ser repensados no ambiente atual, um deles é o livro. Nessa seção, analisa-se o negócio de publicação de livro, e discute-se um novo modelo para esse negócio. A relevância dessa discussão se dá em primeiro lugar porque o assunto impacta a própria forma como essa informação está chegando ao leitor. Em segundo lugar, trata-se de uma ilustração de como se pode e se deve repensar a verdadeira essência de negócios tradicionais, considerando a condição das variáveis de ambiente no momento atual. Se a própria forma como um produz, e comercializa um livro pode e deve ser repensada, que outros negócios também merecem ter seu modelo repensado ? A partir do conhecimento que se tem - ou se pretende ter - de TI, que novos negócios poderão surgir ? De que forma o profissional ou empresa de TI poderá se inserir no novo ambiente aproveitando-se das melhores oportunidades ? O que se deve estar fazendo hoje para estar em boa posição para aproveitar essas oportunidades no futuro ?

2.18.1 O modelo tradicional

Apesar de toda a modernidade, e de toda a impressionante produção de obras de literatura técnica, o modelo de negócios na produção de livros curiosamente pouco evoluiu em relação ao modelo tradicional, que é mais ou menos assim: Um autor tem uma idéia e produz um original. Apresenta esse original a um editor, que é um empreendedor literário. O editor avalia o potencial de venda da obra, e caso se interesse pelo potencial retorno do investimento, banca a edição do livro. Geralmente, profissionais de revisão literária trabalham junto com o editor, e auxiliam o autor a corrigir comuns pequenos erros da obra. O editor faz a impressão numa gráfica própria ou terceirizada, geralmente pela tecnologia *offset*, que requer a produção de chapas. Considerando o custo da produção das chapas, o empreendimento se paga com uma tiragem a partir de aproximadamente 2000 cópias. O gasto mínimo de papel é considerável, e portanto a produção de um livro é um empreendimento em que se arrisca dinheiro sem garantia de retorno. Na contabilidade do empreendimento há que se considerar também o risco de o produto encalhar (não ser vendido). O processo ainda requer distribuição e venda dos livros. A distribuição é feita por distribuidores, que entregam os livros a livrarias.

Em resumo os atores do processo estão listados abaixo, com sua participação no processo.

- Autor: tem a idéia original. Produz o original da obra em formato eletrônico.
- Revisor: apoia o autor na reparação de pequenos erros que podem ocorrer na obra original.
- Editor: é um empreendedor literário, que avalia o potencial da obra original e se acreditar nela banca os custos da produção de uma tiragem.
- Gráfica: executa a produção física do livro impresso.
- Distribuidor: distribui os livros da empresa de edição para os pontos de venda.
- Ponto de venda (loja de livros): revende livros.

2.18.2 Mudanças no ambiente

O modelo tradicional de produção de obras literárias baseia-se em algumas características que eram válidas no passado, mas que gradualmente se modificaram. Para se pensar num modelo novo para produção de livro, convém lembrar como eram as características que geraram o modelo tradicional de produção de livros, e comparar com as características do modelo atual.

Características do passado	Características ambiente atual
A única forma economicamente viável de se executar a impressão de um livro é por meio de grandes impressoras industriais, que tem custo proibitivo para pequenos empreendedores. Mesmo usando esse tipo de impressora, o custo somente torna-se adequado com tiragens de livros relativamente grandes, de pelo menos cerca de 2000 exemplares.	Há impressoras por toda parte, tanto de uso pessoal quanto de uso industrial. O preço da impressora caiu muito. Ainda há ganhos substanciais de custo pelo uso de grandes impressoras industriais, mas cada vez mais torna-se competitivo o uso de tecnologia que imprime poucas cópias ao invés de uma grande tiragem. Considerando o risco de uma obra encalhar (e a necessidade de remunerar esse risco), é cada vez mais atrativa a idéia de usar uma tecnologia de impressão que gere tiragens pequenas.
A barreira de entrada para a produção de uma obra literária é grande. Há relativamente poucos livros, e portanto a informação era relativamente escassa. O leitor procurava pela informação avidamente.	Gradualmente tornou-se bastante fácil publicar um livro. Somando esse efeito com o turbilhão de novidades no mercado de tecnologia de informação, ocorreu uma explosão no número de obras literárias no setor. Adicionalmente, a Internet oferece muitíssima informação gratuita (para quem sabe encontrar).

	<p>Nunca consumiu-se tanta informação como no ambiente atual. Mas a informação não é mais escassa, ao contrário. O leitor está inundado com o excesso de oferta.</p> <p>O autor quer disseminar suas idéias, e se interessa por formas inovadoras para convencer o leitor a escolher o seu conteúdo, num contexto em que há inúmeras opções.</p>
Copiar conteúdo requer tempo e trabalho consideráveis.	<p>A tecnologia⁴ tornou extremamente simples copiar praticamente qualquer tipo de conteúdo.</p> <p>Tentativas de barrar legalmente as cópias tem tido sucesso limitado.</p>
O empreendimento de publicar e comercializar um livro era o próprio negócio principal. A missão da editora era proporcionar meios para a realização desse negócio.	<p>O negócio principal, em muitos casos, tornou-se colher frutos indiretos de idéias que se tem. Para isso, as idéias precisam ser disseminadas. Portanto o autor muitas vezes não tem interesse em mecanismos que dificultem a cópia do conteúdo que produziu. Nesses casos, quanto mais o conteúdo for copiado, melhor para o autor.</p> <p>Um exemplo antológico dessa característica é caso do Linux. O autor original da idéia tornou-se conhecido mundialmente. O compilador gratuito de C/C++, pela gnu, é outro exemplo.</p>
A editora executa a missão de promover as obras que publica. A logística de distribuição física é vantagem competitiva.	<p>O autor pode distribuir seu conteúdo pela Internet. O leitor, apoiado pela Internet, é informadíssimo sobre novidades. Quando um conteúdo gratuito é disponibilizado, rapidamente os interessados ficam sabendo. As trocas espontâneas de mensagem informando sobre conteúdos (o que alguns autores chamam de <i>marketing viral</i>) substituem o trabalho formal de promoção da obra.</p> <p>O fato de ser gratuito torna o conteúdo diferenciado, e muitas chances de ganhar a disputa pela atenção do leitor.</p>
O distribuidor cumpre a missão logística essencial que é fazer chegar o conteúdo (o	A distribuição pela Internet, e a autorização para se copiar amplamente o conteúdo propiciam uma distribuição barata e de

⁴ Por exemplo: foto-cópia, scanner, Internet, print-screen, copiador de CD e DVD, software de cópia *peer-to-peer* tipo Napster, *digi*-camera, digitalizadores de vídeo, etc.

livro impresso) onde está o leitor.	abrangência mundial.
O ciclo de atualização conteúdo é longo. A existência de estoque de livros impressos (consequência de tiragens numerosas) é fator de inibição para a atualização freqüente do conteúdo.	Não há impedimentos para revisão periódica do conteúdo gerado com tiragens pouco numerosas; muito menos no caso de formato eletrônico. Na área de tecnologia de informação, caracterizada por ter sempre inúmeras novidades, a possibilidade de revisão periódica do conteúdo é particularmente bem vinda.

2.18.3 O novo modelo

Como deve ser um modelo de negócios adequado ao ambiente conectado da atualidade ? Heis uma pergunta difícil de ser respondida. Vamos tentar responde-la.

O processo começa igual: com o autor tendo uma idéia e produzindo um original. No novo modelo, assume-se que o leitor possui ou consegue usar computadores com acesso a Internet. O autor disponibiliza a obra em página web, e o leitor copia a obra e a imprime se quiser, pagando do próprio bolso os custos de impressão. Não é necessário remunerar o risco de a obra não ser vendida e encalhar, pois a distribuição é eletrônica e a impressão é feita localmente ou por empresas que usam tecnologia de tiragem pouco numerosa.

No primeiro momento as editoras, gráficas, distribuidoras e lojas de livros tendem a ter reação negativa a qualquer novo modelo. Isso porque a tendência é que suas atividades sejam consideradas irrelevantes no novo cenário. Portanto, supostamente, esses atores correm o risco de que suas atividades serem substituídas por um modelo em que não participam.

Acontece que o mundo sempre mudou, e tudo indica que continuará sempre mudando. Várias atividades econômicas nasceram e morreram ao longo da história. Tudo isso é muito natural. E o que a história nos mostra ? Os novos modelos de negócios muitas vezes não eliminam os modelos anteriores, mas convivem com eles. A televisão não acabou com o rádio. A Internet não acabou com os jornais e revistas. O video cassete, a TV por assinatura e o DVD não acabaram com o cinema. Portanto, é bastante razoável assumir que a possibilidade de distribuir gratuitamente um livro eletrônico não seja fator para eliminar a necessidade de editor, gráfica, distribuidor e revendedor.

Com vistas a se definir um novo modelo, em primeiro lugar, deve-se ressaltar que para que um conteúdo seja oferecido gratuitamente, o autor tem que concordar com isso. Por vários motivos, muitos autores poderão não querer fazê-lo. Portanto, haverá muito mercado para as editoras, distribuidoras e vendas de livros.

Outra característica é que o leitor pode querer comprar o livro impresso, mesmo tendo-o em formato eletrônico. Isso porque ler o livro em papel é mais confortável e conveniente que ler na tela. Além disso, geralmente o leitor não tem impressora profissional a disposição. E imprimir grandes volumes em impressora pessoal é caro e inconveniente. Qualquer gráfica – grande ou pequena – pode copiar o conteúdo, imprimir e vender o livro. Portanto o novo modelo faz gerar oportunidade para muitas empresas.

O autor providencia uma “versão oficial” do livro. Essa versão tem atrativos extras em relação ao conteúdo distribuído gratuitamente. Por exemplo: a versão oficial tem capa e encadernação bonitas, pode ter um CD com arquivos de interesse do leitor, ou mesmo conteúdo adicional não distribuído na versão gratuita. A estratégia básica no novo modelo é a seguinte: a partir da distribuição gratuita, aumenta-se grandemente o número de pessoas que tomam conhecimento do conteúdo. A gratuidade do conteúdo é fator essencial de marketing do conteúdo no ambiente ultra-competitivo atual. Mas o conteúdo gratuito e de baixa qualidade não atrairá o leitor. Portanto, o conteúdo precisa ser de alta qualidade para que o leitor efetivamente se motive para efetivamente acreditar na obra e a comprar a versão oficial do livro. Considerando o volume expressivo de pessoas que tomam conhecimento do conteúdo, se uma pequena parcela desse todo decidir comprar a versão oficial, já trata-se de um contingente substancial.

Uma editora parceira do autor se encarrega de vender a versão oficial do livro. Assim, a editora não perde o significado no novo ambiente, apenas adaptou-se. O novo modelo para produção de conteúdo literário não é uma afronta às editoras e demais atores do modelo tradicional, ao contrário. É apenas uma necessária adaptação aos novos tempos. Todo o processo de transição para um novo modelo tenderá a ocorrer gradualmente, pois muitos tipos de conteúdos não são candidatos a serem oferecidos gratuitamente. Os autores e editores nasceram para serem parceiros, e não inimigos. Além disso, o autor tem interesse em medir a penetração de suas idéias. Uma das formas de fazê-lo é contar o número de livros vendidos na versão oficial.

As revendas de livros também não perdem o significado. A editora geralmente dá descontos para revendedores de livros. Esse desconto converte-se na margem financeira que sustenta a atividade de revenda de livros. A questão é que o preço final do livro não pode ser muito mais caro que a versão impressa por motivos óbvios.

Outra dimensão a se explorar no novo modelo para produção de conteúdo literário é o modelo de patrocínio do conteúdo. A idéia de financiar a produção de conteúdo com patrocínio não é nova realmente. O modelo de negócios da televisão é basicamente esse. O mesmo ocorre com muitas revistas e jornais. Páginas na web focadas em conteúdo também usam esse modelo. Não há em princípio qualquer motivo para que um livro não o adote também.

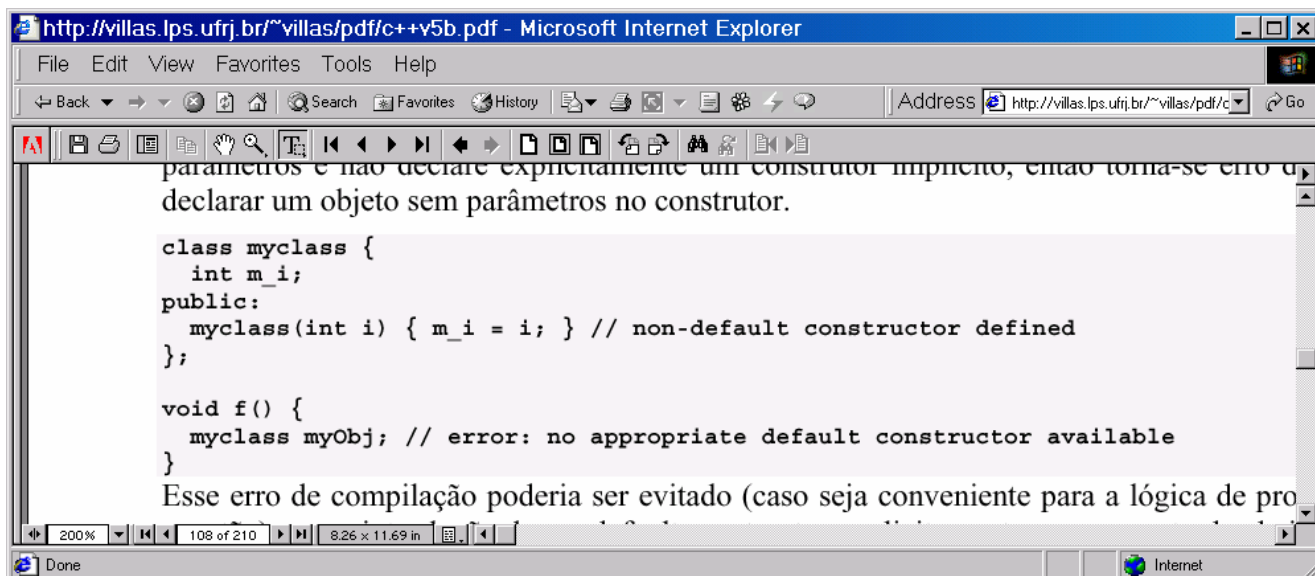
Num modelo de conteúdo financiado por patrocínio, espera-se que o espectador (no caso o leitor) **preste atenção nos patrocinadores do livro, inclusive informando ao patrocinador de que leu a sua mensagem a partir da publicação nesse livro.** Por fazer esse ato simples, o leitor estará reforçando o novo modelo que lhe dá a grande vantagem de disponibilizar conteúdo gratuito. Igualmente importante é que **empresas reconheçam o livro patrocinado como uma forma de divulgar sua mensagem para o público.** Adicionalmente, leitores e patrocinadores devem comunicar-se com o autor sugerindo como esperam ver o conteúdo adaptado nas novas versões que surgirão.

A proposta desse livro é a seguinte: um livro didático destinado a estudantes e profissionais de computação, e assuntos relacionados. O projeto de produção do livro prevê revisão e melhoramento do conteúdo 2 vezes por ano, logo antes do início do período letivo universitário – em Fevereiro e Julho aproximadamente.

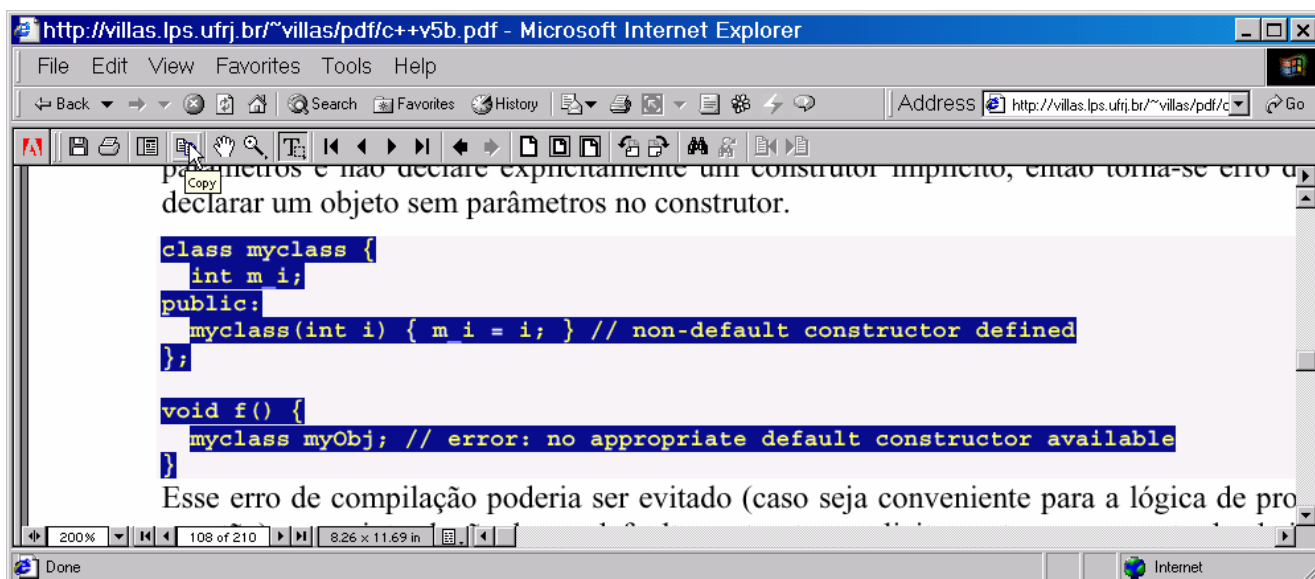
2.18.4 Leitor gosta de ter livro em formato eletrônico

Há valores adicionais, percebidos pelo leitor, pelo fato de um livro ser disponibilizado em formato eletrônico. A existência desses valores é mais um elemento a clamar por um novo modelo para livros. Um desses valores é a possibilidade de se procurar *search* palavras no texto. É bastante comum que se precise procurar uma palavra num livro. Na versão em papel, o melhor que se pode fazer é acrescentar um índice remissivo. Num livro disponibilizado para o leitor em formato eletrônico, a procura eletrônica é possível.

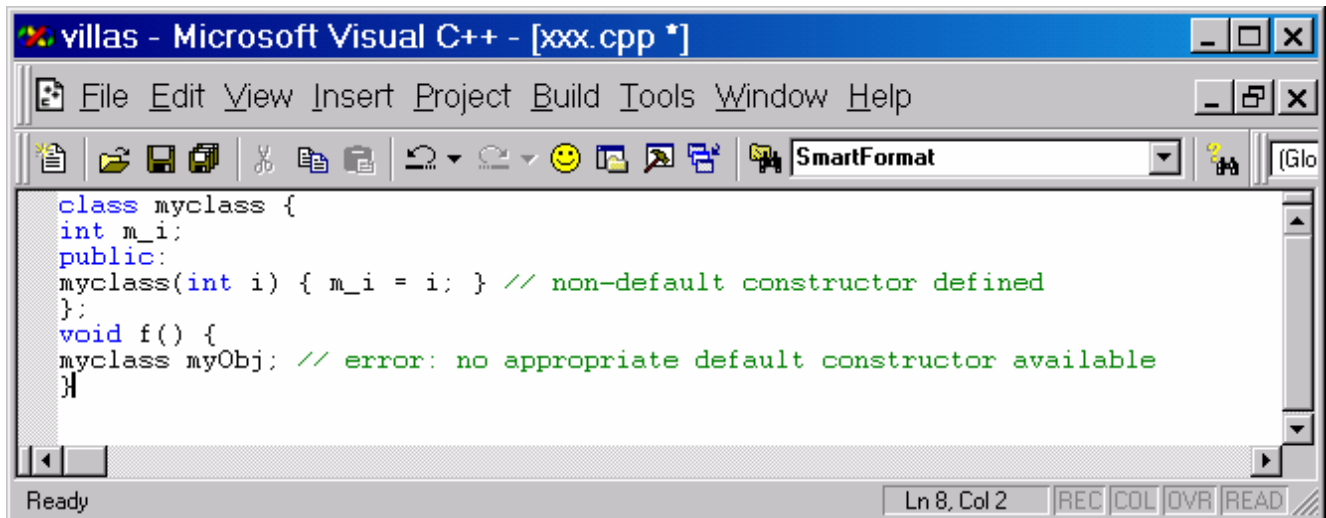
Outro valor é o utilíssimo procedimento corta-e-cola (*cut-and-paste*). Em formato eletrônico, tal como PDF, pode-se cortar e colar trechos de código diretamente do livro eletrônico para o editor de texto do compilador. Para quem não tem experiência com isso, aqui vai uma receita passo a passo sobre como copiar código do livro eletrônico diretamente para seu compilador (ou para outro aplicativo). No exemplo, o código fonte é copiado para o Visual C++. Abre-se o livro na página que se quer copiar, como mostrado abaixo (nesse caso, o PDF reader é aberto por dentro do Internet Explorer, como acontece num texto copiado da web).



Marque o código que gostaria de copiar, e em seguida clique no botão de copiar



Copie o código (Edit – Paste) no Visual C++. A indentação se perde com a cópia.

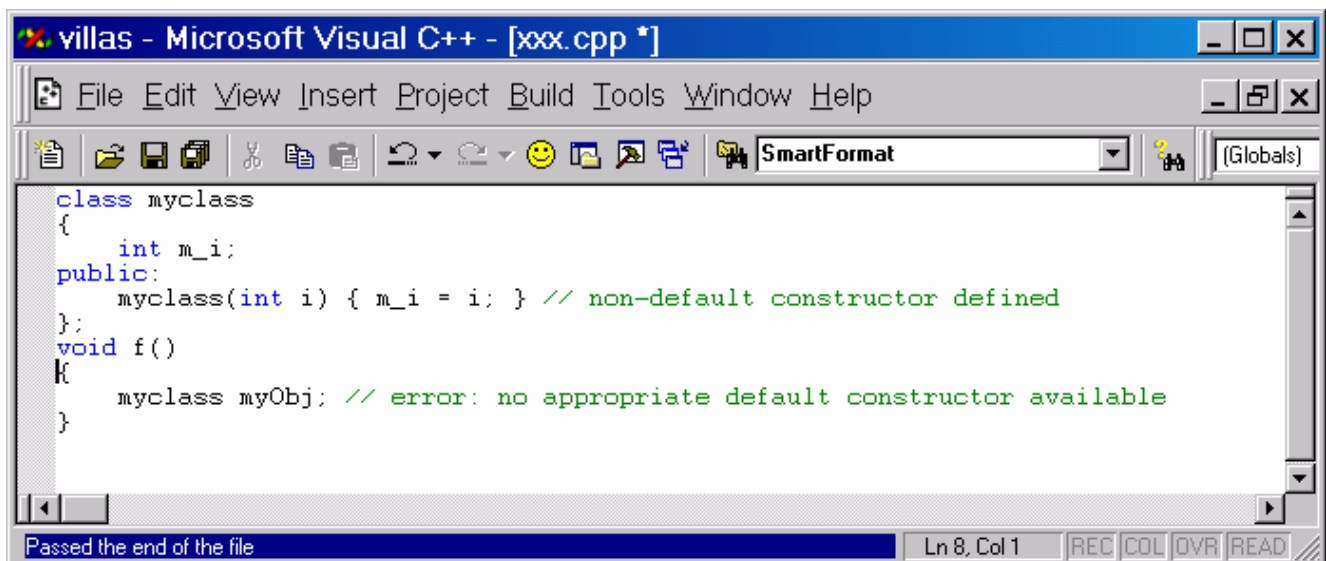


The screenshot shows the Microsoft Visual C++ editor window titled "villas - Microsoft Visual C++ - [xxx.cpp *]". The menu bar includes File, Edit, View, Insert, Project, Build, Tools, Window, and Help. The toolbar contains various icons for file operations and editing. The code editor displays the following C++ code:

```
class myclass {  
int m_i;  
public:  
myclass(int i) { m_i = i; } // non-default constructor defined  
};  
void f() {  
myclass myObj; // error: no appropriate default constructor available  
}
```

The status bar at the bottom indicates "Ready" and "Ln 8, Col 2".

Usando a macro de reformatação de texto (veja seção 4.1.7.1, na página 108), obtém-se o texto no compilador com indentação.



The screenshot shows the same Microsoft Visual C++ editor window after applying a text formatting macro. The code is now properly indented:

```
class myclass  
{  
    int m_i;  
public:  
    myclass(int i) { m_i = i; } // non-default constructor defined  
};  
void f()  
{  
    myclass myObj; // error: no appropriate default constructor available  
}
```

The status bar at the bottom indicates "Passed the end of the file" and "Ln 8, Col 1".

2.18.5 Aferição de mérito acadêmico

A questão da aferição de mérito acadêmico é uma das que precisa ser considerada na definição de um novo modelo para produção de livros didáticos. Existe uma saudável disputa por oportunidades na carreira acadêmica. Calcula-se regularmente uma “figura de mérito” da atuação do docente. A concessão de oportunidades por diversos órgãos e instituições de apoio universitário é feita com base nessa figura de mérito. Dentre as atividades que contam ponto para essa figura de mérito incluem-se a publicação de artigos em conferências e revistas técnicas, nacionais e internacionais. Outra atividade que conta é a publicação de livros.

O problema é que com a queda na barreira de entrada para a publicação de livros, é relativamente simples publicar conteúdo de qualidade duvidosa. Aliás, uma das características atuais é que com a multiplicação explosiva de conteúdo, torna-se difícil decidir onde são as fontes confiáveis. Portanto, o novo modelo, para ser bem sucedido, deverá ser capaz de propor uma medida de qualidade do conteúdo, e com isso permitir a aferição do mérito acadêmico.

Um dos parâmetros de aferição é o número de cópias vendidas da “versão oficial”. Outro parâmetro é o envio de depoimentos de leitores, especialmente de professores e/ou instituições que optarem por adotar o livro em seus cursos regulares.

2.18.6 Quem está usando o modelo novo

Com algumas variações, há bastante gente trabalhando no novo modelo em que disponibiliza-se conteúdo com autorização de cópia. Abaixo, estão alguns exemplos.

- Bruce Eckel é autor de vários livros, incluindo “Think in C”, “Think in C++”, “Think in Java”. Ele oferece gratuitamente a versão eletrônica de seus livros na sua página web [53].
- O famoso livro de “Numerical Recipes” (receitas numéricas), que tem versão em FORTRAN, C e C++, tem seu conteúdo oferecido gratuitamente na web [39].
- A Biblioteca Universal (“The Universal Library”) [60] é um projeto que oferece vários tipos de conteúdo gratuitamente, incluindo livros, apresentações de multimedia, etc.
- O livro ideavirus, por Seth Godin, é um excelente exemplo desse novo modelo. Não só esse livro é disponibilizado gratuitamente (embora o livro exista para ser comprado pela Amazon.com por exemplo), como o assunto do livro é sobre exatamente um novo modelo para disseminação de idéias em geral. Pegue o livro nesse url: <http://www.ideavirus.com/>.
- O sitio apostilando [80] oferece diversos conteúdos em Português para download. Há diversos materiais tipo apostila de linguagens de programação, e tutoriais de interesse de TI.

2.18.7 Idioma Português no mundo

O Português é um dos pouquíssimos idiomas do mundo que se expandiu por vários países. Os países que falam Português no mundo, com sua população aproximada em milhões, estão na tabela abaixo⁵. Esses países possuem uma “Comunidade dos Países de Língua Portuguesa” [46].

⁵ Fonte: www.cplp.org, e enciclopédia Encarta.

País em que se fala Português	População (milhões)
Brasil	160
Portugal	11
Angola	11
Cabo Verde	0,5
Guiné Bissau	1
Moçambique	16
São Tomé e Príncipe	0,1
Timor Leste	1
Macau (atual província da China)	0,4
TOTAL	201

Considerando a população total do mundo como aproximadamente 5 bilhões de habitantes, conclui-se que cerca de 4% da população mundial fala Português.

2.19 Tribos de tecnologia

O uso de uma tecnologia - tal como uma linguagem de computador - geralmente cria uma “tribo” de usuários que gostam dessa tecnologia. Os membros da “tribo” costumam achar que a “sua” linguagem é a melhor. Por isso é que torna-se difícil julgar de forma isenta qual seria a linguagem melhor. A mentalidade de tribo é de certa forma semelhante a mentalidade que une os torcedores de um time de futebol, ou os crentes de uma religião.

Por exemplo: quem está habituado a usar unix (e.g. linux) como servidor, geralmente não tolera usar Windows. Vice versa também acontece. Uma empresa com cultura Windows bem consolidada muitas vezes reluta em usar servidores unix, ainda que a licença do software seja gratuita.

Para quem conhece bem uma certa linguagem de computador, e essa linguagem atende às suas necessidades, a tal linguagem é em princípio a melhor opção. Mas a experiência mostra que o ambiente de trabalho com tecnologia de informação muda constantemente, e muito rapidamente. Uma linguagem de computador não se “deteriora”, no mesmo sentido que acontece com uma máquina a medida que se torna velha. Mas considerando-se a constante mudança do ambiente, se uma linguagem de computador deixa de ter suporte às novidades que o mercado exige, então essa linguagem de certa forma está se deteriorando. Há inúmeros exemplos de linguagens de programação muito boas que entraram em desuso, pois não houve uma “manutenção”. O dBase ou o Clipper para sistema operacional DOS, são um bom exemplo. São duas linguagens que foram muito importantes no seu tempo. Mas o ambiente passou a valorizar o Windows, e as versões para Windows desses programas custaram muito a sair. Além disso, com o crescimento meteórico do tamanho das bases de dados que para as quais se desenvolvem sistemas, era imperioso que houvesse suporte para uso de

hardware mais moderno, tal como unidade de armazenamento maior e mais rápida. Assim, os usuários de dBase e Clipper ficaram de certa forma pressionados a migrar para uma alternativa.

Uma boa tribo de tecnologia (“boa” no sentido de dinâmica, vigorosa) costuma trocar muitas informações entre seus membros. Na era da Internet, essa troca de informações costuma dar-se por web sites, listas de email, etc.

Quem trabalha com recursos humanos para tecnologia deve estar ciente do comportamento de “tribo” que esses profissionais geralmente têm. Não basta oferecer salário (embora isso seja bastante importante). É preciso oferecer para o profissional de tecnologia um ambiente em que ele se desenvolva profissionalmente a partir das tribos a que pertence ou que gostaria de pertencer. Uma empresa do setor de tecnologia de informação que preza e investe em seu futuro e que desenvolve tecnologia quase sempre vive o dilema de “atrair e reter talentos”. São as pessoas que a empresa consegue reunir que fazem o que tem que ser feito. As pessoas (os colaboradores), mais que os equipamentos ou o prédio são o principal ativo da empresa. É interessante observar que pode-se possuir prédios e equipamentos, mas não pessoas. Pode-se apenas atrair pessoas. Por isso, as melhores empresas cuidam de dar aos seus colaboradores (também referidos como “trabalhadores do conhecimento”) um ambiente agradável e adequadamente desafiador do ponto de vista profissional.

2.20 A escolha tecnológica é uma escolha estratégica

Uma escolha estratégica é aquela que potencialmente leva um país, um povo, uma empresa ou um indivíduo a adquirir vantagem competitiva sustentável. Uma escolha estratégica mal feita pode levar a perda de tempo e dinheiro em grandes proporções, para dizer o mínimo. Uma escolha bem feita pode levar ao oposto, isto é, a poupar tempo e dinheiro. A escolha de tecnologia é uma escolha estratégica. Por escolher uma tecnologia, se está criando ligações duradouras com certos tipos de fornecedores. Se essas ligações colocam-nos em posição ruim de negociação para obter suprimentos indispensáveis, a escolha pela tecnologia em questão pode ser inadequada do ponto de vista estratégico. Por outro lado, se uma opção nos leva a uma posição em que podemos negociar bem com fornecedores, e também bem com clientes, então essa opção leva a vantagem competitiva sustentável, e portanto a ser bem considerada do ponto de vista estratégico.

Escolher um software é em geral uma escolha estratégica. Se o software for o piloto de um projeto que no futuro se imagina expandir grandemente, o preço da licença do software é um parâmetro básico. Se o ambiente que se está montando não requer compra de diversas cópias do software, então é menos problemático optar pelo seu uso. Por exemplo: seja uma empresa que pretende posicionar-se como desenvolvedora de soluções para sistemas na web. Para essa empresa, pode fazer sentido comprar uma licença muito cara de software para

desenvolver soluções com alta eficiência (no desenvolvimento e no desempenho), desde que não seja necessário comprar licenças para o cliente final. Por outro lado, adotar como padrão um software que requer compra de inúmeras licenças é uma decisão estratégica arriscada. Esse é o cerne da discussão em torno de sistema operacional gratuito versus sistema operacional pago. É claro que a longo prazo existe o interesse em se colocar computadores para praticamente todas as atividades humanas. Se cada um desses computadores precisar de uma licença de sistema operacional, será muito caro implementar o novo ambiente. Por outro lado, o sistema pago é em geral mais intuitivo e atrativo ao usuário final que a alternativa gratuita.

Há ainda outras características que podem tornar o software pago melhor que o produto gratuito com o qual concorre diretamente. Uma delas é a responsabilidade pelo software. Quando uma empresa se compromete a fornecer uma solução para um cliente, caso a solução tenha algum problema a responsabilidade é dela. Se uma empresa A compra um software de uma empresa B para fornecer uma solução para um cliente, e se o software tem problema, a firma A tem como cobrar judicialmente se necessário pelo funcionamento do software que comprou, e pode se justificar ao cliente no caso de algum problema, por provar que está acionando a empresa B. Caso o software em questão seja gratuito, a empresa A não tem como acionar ninguém no caso de problemas. Esse “vácuo de responsabilidade” causa incômodo ao gerente técnico que precisa decidir qual software usar para resolver um problema.

Outra característica que importa é o custo total de posse (*TCO - total cost of ownership*) que um sistema tem para quem o adquire. Considerando-se um *mind share* já bastante consolidado do software pago, quem opta por essa solução tem como vantagem adicional o fato de que muitos colaboradores já possuem conhecimento do uso desses softwares. Portanto, não é preciso investir tanto tempo e dinheiro em treinamento.

A escolha de uma linguagem de computador é a escolha de uma tecnologia. Essa escolha é estratégica, pois uma escolha errada pode fazer perder tempo e/ou dinheiro. Para tornar-se especializado - em nível senior - numa linguagem de programação é preciso investir tempo. Trabalhar com diversas linguagens de programação pode tornar muito difícil o trabalho de compreender bem todas elas.

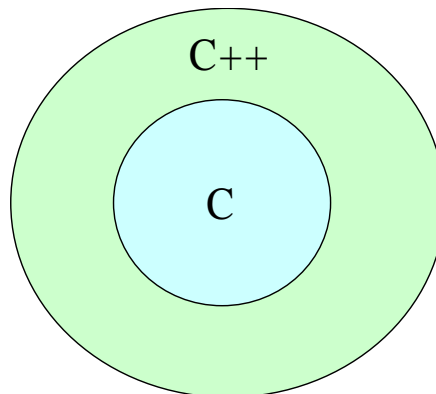


Figura 3: C++ é um super-set de C

As linguagens C / C++ possuem uma filosofia de retirar da própria linguagem tudo o que pode ser escrito em cima dela. Isto quer dizer que muitos comandos que o leitor deverá usar (tais como aqueles necessários para colocar mensagens na tela ou manipular arquivos) e que em outras linguagens correspondem a comandos intrínsecos da linguagem, passarão a ser em C / C++ simples chamadas de funções escritas na própria linguagem. Estas funções, que serão mostradas mais tarde, estão disponíveis na biblioteca padrão da linguagem e podem ser usadas em qualquer compilador C / C++. Esta sutil diferença de filosofia levará a facilitar o trabalho daquele que cria o compilador de uma linguagem de muito poucos comandos onde se compila a biblioteca padrão. O mercado comprova este fato oferecendo ao comprador muito mais opções de compilador C / C++ que qualquer outra linguagem.

2.21 Java × C++

Java é uma linguagem orientada a objetos, feita para ser aprendida facilmente por quem tem conhecimento de C++, sendo mais simples que esta. Além disso, trata-se de uma linguagem que possui um código intermediário – *bytecode* – que pode ser pós compilado para código nativo teoricamente em qualquer ambiente.

Essa linguagem foi proposta inicialmente pela empresa Sun, que conseguiu reunir um importante consórcio de empresas apoiando o padrão.

2.21.1 Porque sim Java

Nessa seção, veja as qualidades de Java, e portanto os motivos para se adotar essa tecnologia.

- Java implementa orientação a objetos muito bem, sendo mais fácil de se aprender que C++ e mais segura (não possui comandos tão poderosos).
- Java é multiplataforma a nível de *bytecode*. Portanto, vale o slogan da linguagem amplamente repetido: “escreva uma vez e execute em qualquer lugar”.

- Há um importante consórcio de empresas que apoiam Java.
- O mercado já comprou a idéia de usar Java. Isso é um fato muito importante. A massa crítica de empresas e profissionais que adotam essa tecnologia é substancial.
- Supostamente, quem adota Java está protegido contra variações que afetam o mercado de tecnologia ao longo do tempo. Isso porque é sempre possível (teoricamente) fazer uma máquina virtual Java que torna os *bytecodes* executáveis num computador que ainda nem se inventou ainda.
- Há muita gente pesquisando formas de melhorar o Java, e tem havido progressos importantes em desempenho, compatibilidade, etc.
- Java é a única linguagem em que se pode escrever applets, que são programas executados pelo navegador web.

2.21.2 Por que não Java

Nessa seção, veja as desvantagens de Java, e portanto os motivos para *não* se adotar essa tecnologia.

- Java é simples apenas se comparada a C++. Mas comparado a outras linguagens disponíveis, é bastante complexa. Não é fácil formar bons programadores Java em grande quantidade. Quem precisa contratar profissionais sofre com escassez de gente. Essa escassez se reflete em custos mais altos que as alternativas. Se esse custo mais alto não puder se mostrar valioso por algum motivo, em princípio a adoção de Java é uma opção inadequada.
- Java permite fazer várias coisas, mas não todas as coisas que existem. Não se usa Java (ou é muito improvável usa-lo) para software básico realmente pesado, tal como escrever um *device driver*, um sistema operacional, o processamento de sinais em chip, ou uma nova linguagem de computador (todas essas coisas podem ser feitas com C/C++). Portanto, apesar de todo o investimento que se faz em dominar essa tecnologia, ainda há assuntos valorizados que não se pode atender usando-a.
- Java é gratuito, mas não “gratuito para sempre”. Isto é, a Sun (dona do Java) não divulga o código fonte do compilador nem da máquina virtual Java. Já houve caso de conflito de interesses sobre royalties a respeito de Java (a Sun quis cobrar royalties da IBM pelo Java 2). Não é impossível que Java torne-se um produto pago no futuro.
- Muitas vezes há alternativas mais simples para fazer o que Java faz. Quer programar para web (cgi) ? GUI ? Cliente-Servidor ? Tudo isso tem soluções baseadas em tecnologias mais simples que Java. Lembre-se: “simplicidade” traduz-se em custos mais baixos para quem contrata.

- Java possui limitações não resolvíveis de desempenho de execução. Isso torna essa tecnologia não recomendada para aplicações com requisição severa de desempenho de execução (tal como controle de motores, reconhecimento de voz, computação científica, ou outra aplicação que requeira muita potência computacional). Mesmo que se compile Java para código nativo (o que melhora seu desempenho de execução), ainda há que necessariamente se executar o código final num ambiente de múltiplas *threads*, sendo uma *thread* para coletar o “lixo” (memória alocada e que não é mais necessária). É muito elegante a forma como se aloca memória em Java sem a preocupação de libera-la. Mas o preço que se paga por isso é em perda de desempenho de execução.
- Argumenta-se que o problema de desempenho de Java tende a ser solucionado implicitamente, pois os computadores melhoram seu desempenho com o tempo. Ou seja, se uma aplicação em Java está hoje com sendo executada com problemas de tempo de execução, pode-se resolver esse problema com um computador mais rápido, que é lançado no mercado a toda hora. Acontece que a expectativa sobre o uso dos computadores cresce na mesma medida de seu desempenho. Ou seja, quando um computador mais rápido é disponibilizado no mercado, a expectativa de desempenho cresceu também. Há sempre demanda para uso de computadores no limite de desempenho. Portanto, as limitações de desempenho proporcionadas pela linguagem Java custam muito a parar de incomodar.
- Java não conseguiu um consenso de TODOS os atores importantes do cenário. A Microsoft não apoia Java, e esse fato isoladamente é um fator negativo substancial contra o uso de Java. Embora haja muito apoio hoje para Java, o desenvolvimento do mercado sem o apoio de atores muito importantes poderá comprometer o futuro da tecnologia. Na oportunidade, convém reportar que a Microsoft está lançando uma plataforma nova, chamada de .Net, que possui código intermediário (da mesma forma que Java), que pode ser gerado por diversas linguagens, tanto da própria Microsoft quanto de outras empresas. Há várias empresas apostando no novo padrão, inclusive Oracle, IBM e outros parceiros da Sun na linguagem Java. A própria Sun não entrou nessa iniciativa.
- Embora Java seja a única linguagem para se escrever applets, como a Microsoft está muito bem posicionada como supridora de navegadores, e não apoia Java, a máquina virtual Java não vem mais pré instalada no sistema Windows. A máquina virtual não é pré instalada no sistema operacional de uso dominante. É sempre possível instalar “na mão” a máquina virtual, mas usuários menos especializados em geral se atrapalham ao tentar fazê-lo. Além disso, os applets são um tanto

“pesados” para serem usados em páginas web de aplicação geral, já que ainda há muitos usuários que conectam-se à web usando modem, a velocidades de transferência de dados menores que 56K (velocidade relativamente pequena).

Um nicho de mercado para uso de applet são os sistemas corporativos com tecnologia de web. Nesse ambiente, em geral o usuário final é conhecido e pode-se apoiá-lo a instalar a máquina virtual se necessário.

2.21.3 Porque sim C++

Nessa seção, veja as vantagens de C++, e portanto os motivos para se adotar essa tecnologia.

- C++ é reconhecidamente a linguagem mais poderosa do mercado, e a opção de escolha da maioria dos produtores de tecnologia.
- C++ é uma linguagem cujo compilador é relativamente fácil de se desenvolver. Destaca-se o fato de que a gnu oferece um excelente compilador C/C++ com fonte – em C – incluído. Trata-se de uma vantagem indireta, porém muito relevante. Há compilador C/C++ para praticamente qualquer computador ou dispositivo eletrônico de interesse.
- Trata-se de uma linguagem ao mesmo tempo de alto e baixo nível. Tem-se todo o poder de uma linguagem de baixo nível, e pode-se usar a orientação a objetos para aumentar indefinidamente o nível de programação.
- Há uma quantidade gigantesca de bibliotecas e componentes aproveitáveis para aplicações em C++.
- É uma linguagem de sólida tradição. Destaca-se o fato de que o unix foi feito em C. Portanto trata-se de uma linguagem que pode ser aplicada em projetos de alta responsabilidade.
- Quase todos os segmentos de TI permitem uso de C/C++. Não há um “dono” desta linguagem. Há vários sistemas operacionais e compiladores 100% gratuitos que usam o padrão C/C++ (o exemplo mais visível é o Linux). Isso significa na prática que essa linguagem tem inúmeros patrocinadores, famosos e anônimos. Isso mantém o suporte sempre muito atualizado e amplamente disponível na Internet.
- Por usar C++, e compreender diversos componentes diferentes, pode-se eventualmente inventar um novo conceito de aplicativo. Descobrir e explorar um novo nicho de mercado é uma possibilidade muito promissora para o mercado de TI.
- Quem tem interesse em eventualmente mudar de área dentro de TI, aproveita muito a bagagem de uso de C++. Por exemplo: Seja o caso de quem programou para web por algum tempo usando C++, e depois se

interessou por outra área, digamos: reconhecimento de voz. Esse profissional aproveita bem a bagagem anterior. Quem programou para web com uma tecnologia específica para web (por exemplo asp, php) não terá aplicação da bagagem quanto estiver atuando para a nova área. O profissional retornará a condição de principiante (algo terrível para quem trabalha em TI).

- Quem se interessa por eventualmente valer-se de software gratuito como diferencial competitivo, tem grandes vantagens por usar C++. E pode fazê-lo sem dispensar o uso de software pago, pois C++ também se usa com software pago.

2.21.4 Por que não C++

Nessa seção, veja as des-vantagens de C++, e portanto os motivos para se *não* adotar essa tecnologia.

- C++ é linguagem muito extensa (demora-se a aprendê-la) e relativamente desprotegida (é possível cometer erros). Quem usa essa linguagem deve ter habilidade e interesse em eventualmente enfrentar a depuração bastante complexa de um sistema.
- Como aprender C++ é demorado e caro, a hora do profissional é tipicamente mais cara que a alternativa. Um gerente de tecnologia pode dizer algo como “a única diferença entre C++ e delphi é que os profissionais de C++ custam mais caro”. A frase é quase correta. O que esse gerente quer dizer provavelmente é “para o sistema que preciso desenvolver agora, o resultado final de delphi e C++ é o mesmo, e como o pessoal de C++ custa mais caro, é mais vantajoso usar C++”. A questão é que a tecnologia iguala as empresas. Como delphi é mais fácil que C++, é improvável que se consiga fazer algo que se diferencie pela tecnologia usando delphi. Se o diferencial não é tecnologia, algum outro fator terá que se diferenciador, pois do contrário a concorrência tenderá a vencer o produto em questão, ou forçará que seja vendido por muito pouco.
- Quem não tem atitude empreendedora nem de produtor de tecnologia não ganha muito por usar C++. Pode ser melhor usar uma linguagem ou ferramenta mais fácil, produzida por outra empresa. Ainda uma vez ressalte-se que essa opção estratégica pode levar a falta de diferenciação no futuro.
- Quem não tem interesse em eventualmente mudar de área dentro de TI tem ganhos de produtividade por especializar-se numa ferramenta desenvolvida especificamente para sua área. Por exemplo: programar para Windows em Visual Basic é muito mais fácil que com C++. Programar para web é muito mais fácil com asp que com C++. Programação científica é muito mais fácil em Matlab que com C++. O problema é que uma das

tecnologias não se aplica para outra área. Além disso, o preço da ferramenta pode ser muito cara.

- Quem não tem interesse em software gratuito ganha pouco por usar uma linguagem mais difícil de aprender, tal como C++.

2.22 Estratégia para integradores de solução

Os integradores de solução precisam analisar os problemas dos consumidores de TI, e escolher dentre os produtos disponíveis aqueles que melhor se adequam para solucionar o problema. **Essa escolha tecnológica é uma decisão estratégica.**

Quando são lançados novos produtos em TI (tipicamente software), todo o mercado é atingido praticamente ao mesmo tempo. Adotar o novo produto (pode ser nova versão de um produto já existente), não é fator de diferenciação para a empresa que o adquire, ainda que suas qualidades sejam extremamente boas. Uma empresa integradora de solução, ou mesmo um consumidor final de TI, não obtém vantagem competitiva sustentável em relação a suas concorrentes por adotar o novo produto.

Uma boa estratégia a seguir para empresas integradoras de solução é cultivar capital de relacionamento, isto é, bons contatos com o mercado. Com bons relacionamentos, e também tendo-se boa competência técnica, consegue-se bons contratos e a empresa vive. Os contatos são muitas vezes informações pessoais, de difícil obtenção. Portanto um capital de relacionamento sólido é uma vantagem competitiva. O problema é que não há fórmula fácil para se avançar sobre esse terreno. Em outras palavras: a mobilidade do capital de relacionamento é baixa. Quem já o tem, preserva-o, já que trata-se de fonte de vantagem competitiva. Se os contratos estão garantidos pelo capital de relacionamento, a tecnologia em si é praticamente uma mercadoria. Tecnologia é “qualquer coisa não muito cara, que se sabe usar, e que funcione”. Nesses casos, o programador é muitas vezes um profissional de relativamente pouco *glamour*, que apenas deve efetuar seu trabalho sem bugs e sem agregar muito custo. Trata-se portanto de uma estratégia boa para o dono da empresa de integração, mas ruim para o programador, que tem dificuldade de negociar boas remunerações para si.

Para sustentar essa estratégia, as empresas produtoras de tecnologia procuram oferecer produtos que levem as empresas integradoras de solução a ter “custos baixos” (entenda-se: possibilidade de trabalhar contratando profissionais baratos). O mercado cunhou até um termo para referir-se ao fato, que é o TCO (*total cost of ownership*, ou custo total de posse). Se um produto produz bem, e é possível obter resultados a partir de operadores de baixo salário, então trata-se de um produto de bom valor. Nesse caso, novamente chama-se a atenção, o programador geralmente tem dificuldade de negociar boas remunerações. Sua

atividade agrega relativamente pouco conhecimento, e a diferenciação que esse profissional tem de seus concorrentes é pequena.

Que estratégia o programador deve então adotar ? E que estratégia deve adotar uma empresa integradora de solução que gostaria de diferenciar-se pelo uso de tecnologia e não por ter capital de relacionamento ? Seja a estratégia resumida pela frase abaixo.

Atuar com espírito de produtor de tecnologia.

É muito difícil a princípio atuar no segmento de produtor de tecnologia. Um dos motivos para isso é que o conhecimento necessário para tal é muito grande. Para que um dia se possa tentar uma atividade nesse segmento, que é muito bem remunerado, é preciso portanto muito conhecimento de TI. A linguagem de programação mais usada pelos produtores de tecnologia é C++. Isso porque trata-se da linguagem mais poderosa do mercado, sendo ao mesmo tempo de baixo nível e de alto nível. Por ser de baixo nível, há grande poder (pode-se fazer virtualmente qualquer coisa com C++). Por ser de alto nível e orientada a objetos, pode-se desenvolver classes que elevam o nível de utilização da linguagem ilimitadamente. Há diversos produtos importantes para o mercado de TI que foram (são) feitos em C/C++, por exemplo: Linux, Netscape, MySQL, Apache, qmail, etc. O estudo do código fonte desses programas é uma das fontes de conhecimento especializado para produtores de tecnologia.

A realidade do mercado é muitas vezes a solicitação de um cliente para sistema relativamente simples para web ou para Windows. Desenvolver sistemas para web/cgi em geral é algo mais simples que desenvolver sistemas para Windows.

Não se pode possuir pessoas (colaboradores). Pode-se apenas atraí-los. Uma empresa que tenha em sua vantagem competitiva conhecimento técnico altamente especializado, tende a envolver os profissionais mais qualificados nos resultados da empresa. Em outras palavras, os profissionais que atuam diretamente no desenvolvimento de TI de alta qualificação em geral devem ser os próprios donos da empresa, ou ter acesso fácil ao dono, e também ter participar dos lucros. Como o dono da empresa em princípio tem interesse em discussões estratégicas para a empresa, as discussões sobre escolha de tecnologia são estratégicas.

O integrador de tecnologia em geral procura diferenciar-se da concorrência por desenvolver parcerias estratégicas (capital de relacionamento). Isto é, por conhecer pessoas chave de um determinado mercado obtem-se vantagem

competitiva. Outra alternativa é procurar diferenciar-se por conhecimento de tecnologia. Acontece que os produtores de tecnologia, quando concluem um novo produto (a toda hora) o disponibilizam ao mesmo tempo para todo o mercado. Em outras palavras, não é estratégia sustentável querer diferenciar-se da concorrência por utilizar-se de uma nova ferramenta recém disponibilizada no mercado. Eventualmente pode-se obter vantagem competitiva temporária por aprender a usar o novo produto de TI mais rápido que a média do mercado. Mas se o tal novo produto é de fato importante, não tardará e inúmeros concorrentes o estarão usando.

Os produtores de tecnologia em grande parte em geral usam a linguagem C++, porque essa é a linguagem mais poderosa do mercado. Uma estratégia para integradores de solução de diferenciar-se como integrador de solução é atuar com espírito de produtor de tecnologia, isto é, com a linguagem C++.

Há questões que somente podem ser avaliadas com sabedoria se forem analisadas ao mesmo tempo sob o prisma de estratégia e sob o prisma tecnológico. Por exemplo: qual é a melhor tecnologia para desenvolvimento de sistemas na web/cgi ? Há várias opções no mercado, cada qual com vantagens e desvantagens. Veja uma discussão específica sobre isso na seção 14.3, na página 291. Uma das finalidades de se estudar negócios e estratégia num material tutorial sobre TI é justamente dar subsídio para responder a esse tipo de questão.

Há muitas opções tecnológicas, cada qual com suas características de custo, qualidade, disponibilidade, suporte, etc. O profissional de TI deve estudar as opções existentes, e idealmente ser capaz de escolher a opção que melhor se adapta ao problema em questão. A quantidade de conhecimentos tecnológicos é fabulosamente grande, e ainda por cima cresce rapidamente com o tempo. O ambiente profissional muda a toda hora. Por isso, é particularmente adequado pensar na carreira pessoal e na missão da empresa com pensamento estratégico.

Muitas referências se dedicam a explorar como devem ser os negócios dos consumidores de produtos de TI (lojas, escritórios, escolas, etc). Nesse texto, tenho a intenção de discutir como deve ser a estratégia do profissional ou empresa de TI, que tem por missão prestar soluções para esses negócios. Como a estratégia do supridor de TI é muito relacionada às características dos negócios dos clientes de TI, é preciso entender de negócios em geral.

A falta de objetivos estratégicos significa ser conduzidos pelos fatos., e portanto ausência de posição de liderança. Atingir a liderança

Os objetivos mais valiosos são aqueles que garantem vantagem competitiva sustentável.

2.22.1 Análise SWOT

Uma técnica muito usada para avaliar uma proposta de negócio, ou uma estratégia é a análise SWOT, que avalia forças, fraquezas, oportunidades e ameaças. Nessa seção, discute-se a análise SWOT da estratégia “integrador de solução com espírito de produtor de conhecimento”.

- **Forças (*Strengths*)** – criatividade, flexibilidade.

O povo brasileiro é reconhecidamente criativo e flexível. A existência dessas qualidades num profissional de desenvolvimento de software é fator de grande valor. Um profissional com pequena criatividade não inventa nada de novo, e apenas copia soluções inventadas por outros. A falta de flexibilidade leva o profissional a ficar paralisado diante qualquer dificuldade ou incompatibilidade. O ambiente complexo de atuação profissional em TI propicia surgimento de diversas situações em que dificuldades e incompatibilidades aparecem. Nesses casos, é preciso flexibilidade e atitude pro-ativa para propor soluções para o problema.

- **Fraquezas (*Weaknesses*)** – falta de disciplina, falta de método, vergonha de ganhar dinheiro, falta de tradição nesse mercado.

Se comparado a estrangeiros, muitos brasileiros tem pouca disciplina. A falta de pontualidade é um aspecto desse valor cultural. Para conseguir certo tipo de contratos estratégicos e de muito valor, é preciso que a equipe tenha disciplina. A falta de método é outra fraqueza. É relativamente difícil trabalhar com planos de longo prazo. Pode haver dificuldades para se conseguir uma boa discussão sobre estratégia. Há também falta de tradição no mercado de software, dominado por grandes empresas estrangeiras. Muitas vezes o consumidor de TI tem padrões mentais de fornecimento de soluções por empresas estrangeiras, e nem considera as opções nacionais, numa espécie de “protecionismo ao contrário”.

Há um valor cultural brasileiro peculiar, que merece ser analisado ser analisado em separado. É a “vergonha de ganhar dinheiro”. Esse aspecto da cultura é registrado em diversas referências. Apenas para citar uma referência recente, na página 95 da revista quinzenal “Exame” edição 763 de 03 de Abril de 2002, Cláudio Gradilone escreveu um artigo intitulado “Dinheiro não morde”. Heis um trecho do artigo.

O brasileiro dissocia a riqueza financeira da espiritual, como se o fato de ter dinheiro corrompesse as pessoas. No imaginário coletivo, os pobres são felizes e virtuosos, e os ricos são problemáticos.

Como se disse, a tecnologia é forjada pela visão de negócios. Para ser forte em tecnologia de informação, deve-se ter empresas fortes, chefiadas por quem não tem vergonha de ganhar dinheiro. Será que Bill Gates, Steve Jobs, Larry Ellison e tantos outros empresários de sucesso do setor de TI tem alguma vergonha de ganhar dinheiro ?

- **Oportunidades (*Oportunities*)** – O modelo econômico que se usou até agora para implantação de soluções em TI foi em grande parte baseado em soluções proprietárias. Contudo há exemplos gloriosos de utilização de software gratuito. Apenas para citar um exemplo, diversas pesquisas indicam que o servidor web mais usado do mundo, com mais de 60% de preferência é o Apache (software gratuito distribuído com o fonte, com versão para Unix e Windows).

A história daqui para frente pode ser diferente do que ocorreu até agora. A parte mais educada e rica da sociedade já está informatizada. Para atingir uma parcela menos educada e abastada, o modelo mais indicado pode ser diferente, inclusive porque o software gratuito está atingindo níveis de maturidade e qualidade muito bons.

Esse pode ser o ambiente ideal para que empresas menores possam crescer, integrando soluções em TI a partir de componentes baseados em software gratuito. Dessa forma, a solução final fica mais barata que a das grandes empresas. Se a repressão ao uso de cópias ilegais de software aumentar, a oportunidade para as empresas menores pode tornar-se ainda maior.

- **Ameaças (*Threats*)** – A principal ameaça é a concorrência feroz que se tem que travar o tempo todo, com empresas que podem ser muito maiores.

O assunto do título da seção é abrangente, e pode ser explorado por vários ângulos. Nesse texto, se fará discussão do tema focando especificamente características estratégicas - do ponto de vista do indivíduo, e do ponto de vista das empresa - de se decidir investir no estudo da linguagem C++.

O mundo que vivemos é complexo e possui quantidade extremamente grande de informações. A velocidade de aumento da complexidade do mundo é crescente. O ambiente profissional no ramo de tecnologia de informação muda constantemente, de forma cada vez mais veloz. Muitas tecnologias nasceram e ficaram obsoletas rapidamente. Há um considerável stress para quem pretende

manter-se atualizado nesse ambiente, com capacidade de tomar decisões baseadas na avaliação racional das possibilidades disponíveis.

É curioso comentar que todo o vigoroso ambiente de tecnologia se dá baseado nos cérebros humanos, que não melhoram com o tempo. Os humanos de hoje basicamente tem o mesmo número de neurônios no cérebro que os remotos antepassados homens das cavernas . . .

Profissionais e empresas de tecnologia de informação convivem com um problema básico: que estratégia devem adotar ? Antes uma pergunta ainda mais fundamental: Qual exatamente é o objetivo de estratégia que se pretenda adotar ? Uma estratégia adequada deverá levar a:

- Manter o profissional atualizado nos conhecimentos valorizados no mercado, e portanto com boa empregabilidade.
- Levar a empresa a ter uma fonte de remuneração que garanta a sua sobrevivência e crescimento.
- Manter a empresa e o profissional com capacidade de desenvolver produtos de alto valor agregado.
- Maximizar o efeito positivo do tempo investido em estudo. Em outras palavras: escolher tecnologias (e portanto escolher temas de estudo) que levem ao máximo de empregabilidade, valorização profissional, etc. no contexto de uma carreira, profissional ou empresarial, que se pretende consolidar. Note a visão de longo prazo aqui.

A conclusão de qual estratégia se deve adotar virá com a consideração de mais alguns pontos, como discutido a seguir.

2.23 Resumo

A linguagem C++ é reconhecidamente a linguagem mais poderosa do mercado. É a opção de escolha da maioria dos produtores de tecnologia. Trata-se de uma linguagem ao mesmo tempo de baixo e de alto nível. Portanto, pode-se usa-la para implementar componentes de programação a nível de serviço de rede, ou a nível de device driver. Ao mesmo tempo pode-se usar a própria linguagem e desenvolver classes que fazem aumentar o nível de programação arbitrariamente. Além disso, está bastante madura, inclusive com disponibilidade de diversos compiladores - pagos e gratuitos.

C++ é uma linguagem orientada a objetos. Pode-se fazer aumentar o nível de programação arbitrariamente, obtendo gradativamente bibliotecas de componentes reutilizáveis, e com isso obtendo-se alto desempenho de programação. Para quem gerencia projetos de software, a orientação a objetos significa maior reutilização de código e portanto custos mais baixos, tempos de desenvolvimento menores e menos bugs no software. A orientação a objetos tem

aplicação ampla, sendo de particular aplicação no desenvolvimento de interfaces e banco de dados.

O fato de C++ ser poderosa é ao mesmo tempo vantagem e desvantagem. A desvantagem é que para atingir tanto poder, a linguagem é complexa e extensa. Por isso, atingir o nível sólido de competência em C++ toma tempo (leia-se custa dinheiro). Na hora de decidir por uma tecnologia a adotar para solucionar um problema, em vários casos, há alternativas eficientes e mais baratas que C++. Nesses casos pode ser difícil convencer um gerente de tecnologia a desenvolver em C++, a menos que haja um outro objetivo que se queira atingir. Para quem gerencia projetos de software, o grande poder de C++ pode ser um problema, pois é possível os programadores “fazerem besteira” com o poder que a linguagem tem. Para se poder gerenciar (controlar) melhor os programadores, há linguagens menos poderosas que facilitam o trabalho de gerência.

Para se atingir nível de guru em C++, é preciso muita experiência. Pode-se fazer uma comparação com piloto de avião. A diferença entre um piloto experiente e um novato são as horas de voo. E essa diferença é reconhecida pelo mercado. Algo parecido ocorre em C++. Um profissional que tem “horas de voo” suficientes em C++ sente-se seguro na linguagem, e pode ser tão produtivo nela quanto em seria numa linguagem feita para ser simples (como Visual Basic, ou ASP). E pelo fato de estar em C++, pode-se desfrutar de todo o poder da linguagem. Por estar em C++, pode-se eventualmente inventar algum conceito novo e valorizado.

Além disso, a vida nos mostra que a habilidade de adaptação é muito útil. Portanto, ao se atender a uma especificação de um cliente usando a linguagem C++, se está caminhando na carreira profissional, enquanto se aumenta as “horas de voo”. Se no futuro existir interesse em mudar de área dentro da tecnologia de informação, as “horas de voo” serão algo valorizado na bagagem.

Seja um exemplo concreto. Deseja-se definir uma linguagem de programação para um desenvolvimento (digamos: mestrado ou doutorado) em um tópico de computação científica (cálculo numérico). Nessa área, há ainda forte influência da linguagem FORTRAN. E não há realmente nada de errado com essa abordagem. A questão é que ao se fazer o desenvolvimento usando FORTRAN, chega-se aos resultados que se espera do ponto de vista de computação científica, mas o profissional em questão aumentou suas “horas de voo” em FORTRAN, e não em C++. Se no futuro houver interesse em se mudar de área para computação gráfica, ou para banco de dados, ou para web, ou para redes de computadores (apenas para dar alguns exemplos), a habilidade em FORTRAN não seria valorizada. Mas no caso de se ter optado por C++, a experiência nessa linguagem com certeza seria muito valorizada.

Por isso, é comum que o profissional “da tribo de C++” tenda a escolher essa linguagem para tudo o que puder. Essa é uma atitude instintiva tanto para

aproveitar os conhecimentos que já se tem, quanto para fazer aumentar conhecimentos num contexto onde há expectativa de se aproveitá-los em futuros projetos.

Além disso, a partir da habilidade em C/C++ que se cultiva, pode-se eventualmente inventar algo novo e muito valorizado. Abaixo há uma lista de projetos de alto impacto para o ambiente de tecnologia de informação. Todos eles foram feitos em C/C++, e quase todos tem o fonte disponível. Imagine um dia participar de um projeto como os da lista abaixo ! Para um dia chegar a essa competência, é importante que se adquira experiência.

- Linux (versão de unix para plataforma PC).
- Apache (servidor web).
- Netscape Navigator (cliente web).
- Napster (primeira ferramenta de compartilhamento de arquivos com a estrutura *peer-to-peer* (par-a-par)).
- Latex (compilador de texto profissional).
- gcc e g++ (compiladores de C++).
- qmail (servidor de email).
- MySQL e PostgreSQL (dois bancos de dados gratuitos).

A lista poderia ser muito maior, pois esses são apenas alguns exemplos. Abaixo há uma lista de alguns temas quentes de pesquisa. A maioria desses projetos requer grande poder de processamento, e usa (ou pode usar) C/C++.

- XML e automação de negócios.
- Sistemas pela web, e pela Internet em geral.
- Tecnologia de voz (síntese de voz, e reconhecimento de voz).
- Processamento paralelo.
- Qualidade de serviço em redes de computadores.
- Segurança em redes de computadores.
- Processamento de sinais para modulação digital em telefonia celular.
- Redes neuronais e sistemas fuzzy.
- Banco de dados orientados a objeto.
- Tecnologia de vídeo (efeitos em vídeo, vídeo digital, edição de vídeo).
- Programar para chips (eletrônica embarcada) Microcontroladores e DSP.

2.24 Estratégia e C++

Voltemos à pergunta do início do capítulo: Porque estudar a linguagem C++ ao invés das alternativas ?

O que se está fazendo é tentar entender o mundo para que se possa definir uma estratégia pessoal/empresarial para tecnologia de informação. É preciso que se exerça uma atividade no mercado, e ser capaz de obter um rendimento bom para essa atividade. Que estratégia tomar ?

Uma estratégia profissional a se seguir é usar a linguagem C++ para diversos projetos, e com isso gradualmente atingir um grau de elevado conhecimento nessa linguagem. Dessa forma, fica-se qualificado para executar projetos de alto valor, que são em geral mais interessantes e melhor remunerados que projetos “de mercado”.

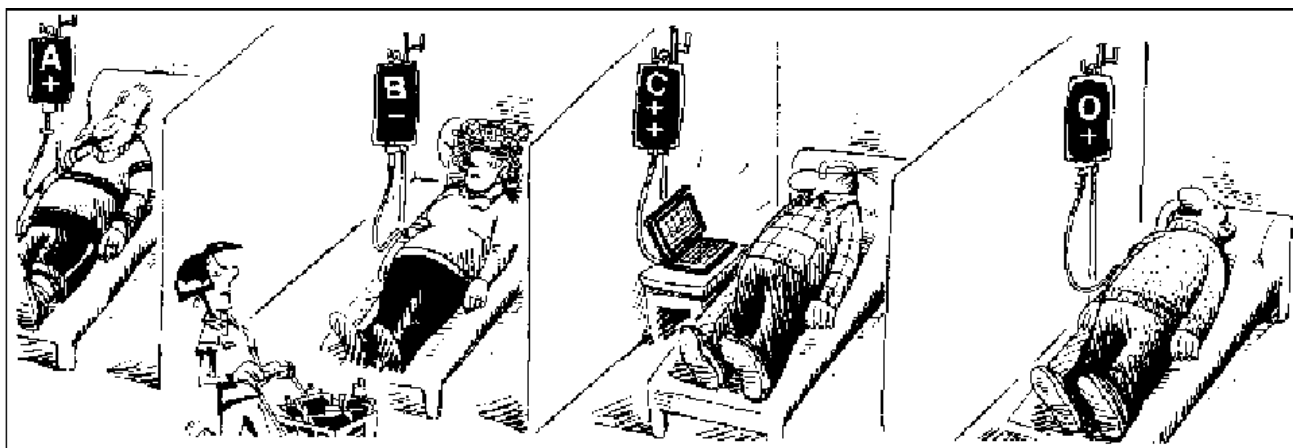
Seja o estudo de caso: desenvolver páginas na web, em html (atividade de webdesigner). Quando ficou claro que a web iria explodir, por volta de 1995, conseguia-se remunerações fabulosas pelo desenvolvimento de páginas web simples. Esse rendimento fez atrair muitos novos entrantes para a atividade. Gradualmente, o grande negócio deixou de ser o desenvolvimento em si das páginas, e passou para ser dar curso para interessados nessa atividade. Adicionalmente, criou-se oportunidade para que se desenvolvessem ferramentas para automatizar e facilitar o desenvolvimento de páginas. O tempo passou, e a evolução ocorreu. Hoje, boa parte do bom dinheiro que se ganha com o desenvolvimento de páginas web está com a venda de ferramentas de apoio ao desenvolvimento, tipo FrontPage, DreamWeaver, GoLive, Photoshop, CorelDraw, etc. Para o profissional ou empresa que se dedicou à atividade de desenvolvimento em si de páginas web, que tem pequena barreira de entrada, a tendência é que surjam novos entrantes no mercado até que a zona de conforto fique muito pequena, ou zero. O mesmo vai ocorrer (já está ocorrendo) com a atividade de dar cursos para profissionais de webdesigner. Em tese, o mesmo ocorreria também com as firmas que se interessaram por desenvolver ferramentas para apoio ao desenvolvimento de páginas web. Acontece, que como se trata de produtos de alto valor agregado de conhecimento, existe uma considerável barreira de entrada para essa atividade. Portanto, a zona de conforto para essa atividade tende a ser maior do que a zona de conforto para as outras atividades relacionadas ao desenvolvimento para web.

Para quem pretende um dia ingressar num time de profissionais muito qualificados que desenvolve software de qualidade, como os que se usa para apoio ao desenvolvimento de páginas web, é MUITO útil ter sólida formação em C++, que é a linguagem que geralmente se usa para esse tipo de programas muito sofisticados.

2.25 Humor

Se todos os argumentos apresentados não foram suficientes para convencer o leitor de que C++ é a uma boa escolha, talvez a charge abaixo, de autoria de Tak Bui e Johnson, possa complementar a argumentação.

Digamos que “C++ é algo que está no sangue . . .”



2.26 Exercícios

- 1) Faça uma pesquisa sobre a pauta de comércio exterior do Brasil, e comente sobre a estratégia do Brasil no cenário de comércio mundial. Dos produtos e serviços que se troca com o exterior, avalie os que tem alto valor agregado e os que não tem. Avalie também os produtos que tem alto conhecimento e os que não tem.
- 2) Comente sobre empresas de sucesso, cuja estratégia você admira.
- 3) Comente o posicionamento estratégico dos negócios abaixo. Avalie a agregação de valor realizado pela própria empresa na composição do valor do produto final. Avalie a capacidade da empresa em diferenciar-se da concorrência, e a barreira de entrada para entrada de concorrentes. Avalie a quantidade e o valor do conhecimento que a empresa acumula com o desenvolvimento de sua atividade econômica.
 - a. Uma loja que vende cartuchos de impressora de marcas famosas, num centro comercial de produtos de informática.
 - b. Uma empresa que vende e instala software de empresas famosas.
 - c. Uma empresa de consultoria em segurança em redes de computação.
 - d. Uma empresa de desenvolvimento de sistemas de informação especializados para aplicações militares, que trabalha em parceria com setores das forças armadas.

- e. Empresa que desenvolve sistemas de informação em redes de computadores, baseado em software de empresas famosas.
 - f. O mesmo caso anterior, baseado em software gratuito.
 - g. Empresa que desenvolve e vende sistemas de reconhecimento e síntese de voz.
 - h. Empresa que revende sistemas de reconhecimento e síntese de voz, desenvolvidos por empresas famosas.
 - i. Empresa que desenvolve sites na web baseado em software de empresas famosas.
 - j. Empresa que desenvolve ferramentas para automatizar o desenvolvimento de conteúdo na web.
 - k. Empresa que comercializa um software de controle de estoque, desenvolvido com uma ferramenta de fácil aprendizado, onde basicamente “arrasta-se blocos e componentes” para o aplicativo e pronto.
 - l. Empresa que dá consultoria em banco de dados, analisando e sugerindo melhorias no modelo de dados do cliente, e adaptando o sistema para diversos produtos de banco de dados disponíveis no mercado.
 - m. Empresa que desenvolve software para ser vendido agregado a um hardware de um parceiro. Por exemplo: desenvolvimento de *device driver* e aplicativo de captura de imagem para um *scanner*.
- 4) Estude a recente plataforma .Net, patrocinada pela Microsoft, com apoio de diversos parceiros. Que tipo de negócios surgem a partir da proposição dessa plataforma ? Dos negócios que se surgirão, quais negócios tendem a ser de alta agregação de valor (*big business*), e quais são de baixa agregação de valor (*small business*) ? Como a sua estratégia, de indivíduo ou de empresa, se encaixa com o uso dessa plataforma ?
 - 5) Na sua opinião, porque a linguagem C++ não é exatamente igual nos diversos compiladores ? E a linguagem SQL (usada em banco de dados relacionais), porque não é 100% igual para todos os produtos de banco de dados ? A tendência é que aos poucos essas linguagens tornem-se 100% compatíveis ?
 - 6) Qual é o futuro do software gratuito ? O linux será gratuito para sempre ? Como os profissionais de tecnologia obterão remuneração num ambiente de software gratuito ?
 - 7) Na época dos mainframes, o software era vendido, mas geralmente havia uma taxa mensal de manutenção. Com a micro informática, iniciou-se uma

era de venda de licenças de software (sem taxa de manutenção). Comente a diferença desses dois modelos de comercialização de software do ponto de vista do fornecedor de tecnologia.

- 8) Cite exemplos de produtos ou serviços que possam ser classificados como “conhecimento empacotado”. Justifique a resposta.

C++ Multiplataforma e Orientação a Objetos

Parte 2: Fundamentos de C / C++

3 Conceitos introdutórios

3.1 Breve história do C/C++

Tudo começou com a linguagem BCPL, por Martin Richards, que rodava no computador DEC PDP-7, com sistema operacional unix (versão da época). Em 1970, Ken Thompson fez algumas melhorias na linguagem BCPL e chamou a nova linguagem de “B”. Em 1972, Dennis Ritchie e Ken Thompson fizeram várias melhorias na linguagem B e para dar um novo nome a linguagem, chamaram-na de “C” (como sucessor de “B”).

A linguagem C não foi um sucesso imediato após a sua criação, ficando restrita a alguns laboratórios. Em 1978 (um ano histórico para os programadores C), é lançado um livro famoso por Brian Kernigham e Dennis Ritchie [4]. Esse livro serviu de tutorial para muita gente e mudou a história da programação em C. De fato, essa primeira versão da linguagem C é até hoje conhecida como “C Kernigham e Ritchie” ou simplesmente “C K&R”.

Em 1981, a IBM lança o IBM PC, iniciando uma família de computadores de muito sucesso. A linguagem C estava no lugar certo e na hora certa para casar-se com o fenômeno dos computadores pessoais. Logo foram surgindo inúmeros compiladores C para PC. Com isso a linguagem C estava livre do seu confinamento inicial no sistema operacional unix.

Conforme a tecnologia de software foi se desenvolvendo, o mercado exigiu que se introduzisse mudanças na linguagem C. Talvez nós devêssemos nos referir as várias versões de C como “C 1.0”, “C 2.0”, etc., mas o mercado geralmente não usa essa nomenclatura. O fato é que modificações importantes foram pouco a pouco sendo introduzidas na linguagem. Uma modificação importante foi a padronização feita pelo *American National Standards Institute* (ANSI), em 1983, que criou uma linguagem conhecida como “C ANSI”.

Entre outras modificações, o C ANSI mudou a forma de se definir um protótipo de função, para que se possa conferir esse protótipo na hora da compilação e acusar erros no caso de se chamar uma função com parâmetros errados. No quadro abaixo é mostrado um exemplo de como se escrevia uma função (fun1) em C K&R e como passou a se escrever em C ANSI. Se segunda função (fun2) chama a primeira com parâmetros errados, o C K&R não tem como saber do

erro na hora da compilação, mas o C ANSI acusa o erro e com isso ajuda o programador a depurar o seu código⁶.

C K&R	C ANSI
<pre>void fun1 (i, k) int i, k; { ... }</pre>	<pre>void fun1 (int i, int k) { ... }</pre>
<pre>void fun2 () { int a1, a2; float b1, b2; /* variáveis locais */ fun1(a1,a2); /* OK em ambas as versões de C */ fun1(b1,b2); /* é um erro, pois os parâmetros estão errados. Mas somente surgirá um erro de compilação no C ANSI, no C K&R não há esse tipo de checagem */ }</pre>	

No C K&R, como não havia como checar se uma função foi chamada com os argumentos certos, não se proibia o uso de identificadores que não previamente declarados. No exemplo acima significa que podia-se chamar a função fun1 a qualquer momento, sem necessidade de declarar o identificador da função. Com o C ANSI, passou a se recomendar (como técnica de boa programação) que toda função fosse declarada antes de ser usada, para que o compilador pudesse checar a chamada da função. A partir do C++ essa recomendação tornou-se uma obrigação, isto é, o compilador simplesmente não aceita compilar um identificador que não tenha sido previamente declarado. Para quem está no começo da aprendizagem, pode ser um tanto difícil compreender agora a diferença entre o C K&R e o C ANSI. O importante é lembrar que existe a versão de C chamada de “C K&R”, que é a versão mais antiga e menos protegida. Posteriormente surgiu o “C ANSI” é mais moderno e mais protegido. O C++ é ainda mais moderno e mais protegido, e além disso incorpora os conceitos de programação orientada a objetos.

⁶ Para quem está iniciando-se na linguagem, possivelmente não entenderá muito bem agora a diferença entre C K&R e C ANSI. Mas a compreensão desse ponto não é indispensável nesse momento. Ou seja, se o leitor não entendeu isso agora, tudo bem.

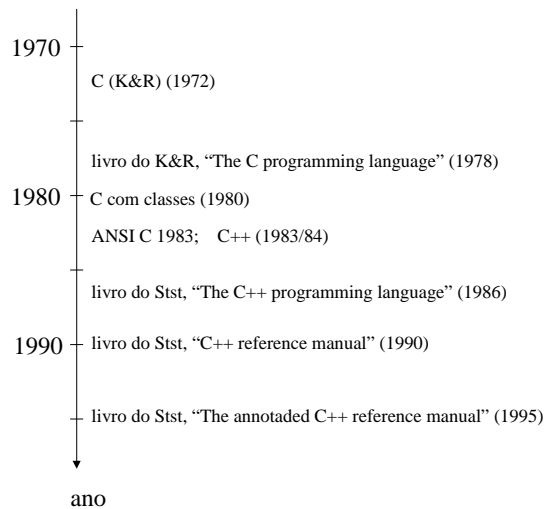


Figura 4: Alguns fatos relevantes na evolução da linguagem C / C++

3.2 Classificação de interfaces de programas

Na informática há hardware e software. O hardware é fisicamente tangível, ao contrário do software. A palavra software significa um programa de computador ou um *sistema*, que é um conjunto de programas de computador.

Há basicamente 3 tipos de interface para programas, como mostrado abaixo em ordem de complexidade (1 = menos complexo, 3 = mais complexo).

1. Console (ou linha de comando), tela de texto tipo *scroll* (rolar). Exemplo: xcopy, grep.
2. Console (ou linha de comando), tela de texto cheia. Exemplo: Lotus 1-2-3, WordStar (para DOS).
3. GUI (*graphics user interface*), também chamado de "interface Windows". Exemplo: sistema operacional Windows e qualquer aplicação para Windows ou o ambiente xWindows do unix.

Uma interface mais complexa agrada mais ao usuário, porém complica a vida do programador. Quando o objetivo é estudar conceitos de programação, a interface é algo que a princípio não está no foco. Portanto, a interface mais adequada para estudo é a interface tipo 1 – console do tipo *scroll*.

Em Windows, é necessário que se abra uma janela de DOS para se ter acesso ao console e poder usar programas com linha de comando (como se fazia na época que não havia Windows). Em unix, há o xWindows, que é um dos tipos de interface gráfica, e também o console de unix.

3.3 Programando para console

Ultimamente, há particular interesse por aplicativos gráficos, com uso intensivo de GUI e multimedia. Há também muito interesse em programação para Internet em geral, e web particular. Para quem está em fase de aprendizado, é importante que se tenha uma base sólida de programação em console antes de se pretender programar em ambientes mais complexos.

Programar para console é a abordagem que historicamente surgiu primeiro que a programação GUI. Programas para console NÃO estão caindo em desuso, apesar da popularidade dos ambientes gráficos. Um exemplo importante de como a programação para console é muito presente na modernidade do mercado de informática, é o caso de programação para Internet com interface CGI. Outro exemplo é a programação para dispositivos programáveis como DSPs, microcontroladores, etc.

Mesmo para quem pretende programar para GUI (como o Windows ou wxWindows), é extremamente importante saber programar para console⁷. Fazendo programas para console que se consegue-se depurar trechos de programa facilmente, e com isso pode-se produzir componentes de software que serão aproveitados em programas quaisquer, incluindo programas gráficos.

3.4 Linguagens de programação de computador

Um computador é acionado por um processador central, conhecido como CPU (*central processing unit*). A CPU é um circuito eletrônico programável, de propósito genérico, que executa instruções em um código próprio, chamado de “código de máquina”. Esse código é fisicamente uma seqüência de bytes⁸. Esse conjunto de bytes é conhecido como programa executável de computador. “Programa” porque é uma seqüência de instruções; “executável”, porque essa seqüência está na forma binária já traduzida para o código de máquina da CPU, portanto está pronto para ser executado (um programa escrito em arquivo de texto não está pronto para ser executado).

Quase sempre, usa-se um computador como um sistema operacional (Windows, Linux, etc.). Esse sistema operacional possibilita por exemplo que se trabalhe com arquivos em disco. Quando um usuário comanda a execução de um programa, o que acontece fisicamente (de forma resumida) é que o sistema operacional copia para a memória o arquivo com o programa executável e pula para o seu início.

⁷ Uma questão interessante é definir qual estratégia adotar para desenvolvimento de programas GUI multiplataforma com C++ (assunto que não é do alcance desse livro). Registre-se que minha opção pessoal para essa finalidade é o uso da biblioteca wxWindows (www.wxWindows.org).

⁸ Um byte é um conjunto de 8 bits.

A CPU somente executa código de máquina binário. É muito difícil para um programador trabalhar diretamente com a informação em binário. Portanto, a técnica básica de programação consiste em utilizar uma “linguagem de programação”. Ou seja, o programador escreve um arquivo de texto (ou um conjunto de arquivos texto) chamado de programa fonte. Em seguida converte esses arquivos texto no código de máquina num processo chamado de “construção” (*build*)⁹

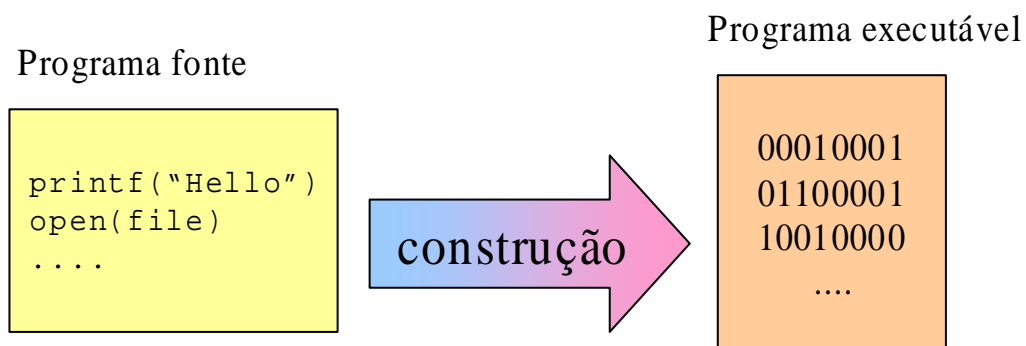


Figura 5: Visão gráfica da construção de um programa

⁹ Muitos autores chamam esse processo de “compilação”, mas evito usar esse termo aqui para que se possa definir com precisão a “construção” de um programa em duas etapas: “compilação” e “ligação”(link), como será explicado.

4 Conheça o Seu Compilador

Esse capítulo explica como conhecer o compilador que se vai usar. Cada seção fala de um compilador diferente. Portanto use esse capítulo assim: verifique dentre os compiladores abordados qual é o que você vai usar e leia essa seção. As demais seções podem ser ignoradas (a princípio), pois falam de outros compiladores.

Para um compilador qualquer, é importante que se adquira as seguintes habilidades:

1. Fazer um programa tipo “hello world” para console.
2. Adicionar argumentos na linha de comando.
3. Usar o help do programa.
4. Usar “projetos” (fazer programas com múltiplos fontes).
5. Usar bibliotecas (*libraries*), isto é,
 - Incluir uma biblioteca num projeto.
 - Fazer uma biblioteca para uma terceira pessoa.
 - Examinar (listar) o conteúdo de uma biblioteca qualquer.
6. Usar ferramenta de debug.
7. Definir um identificador para permitir compilação condicional.
8. Escrever um código para detectar vazamento de memória

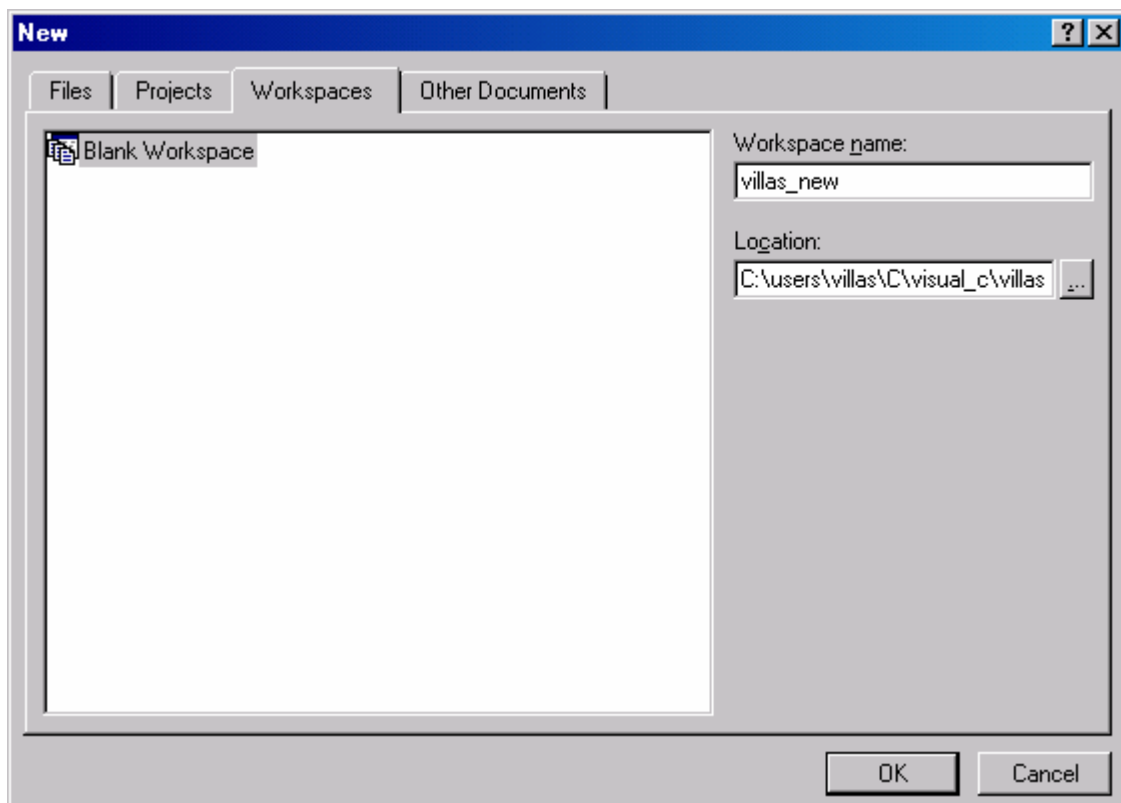
4.1 Visual C++ 6.0 SP5

SP5 significa o “service pack versão 5”, que é distribuído gratuitamente pelo site da Microsoft. Um “service pack” é um “patch”, isto é, uma correção para bugs que é lançado após a versão inicial do produto propriamente dito.

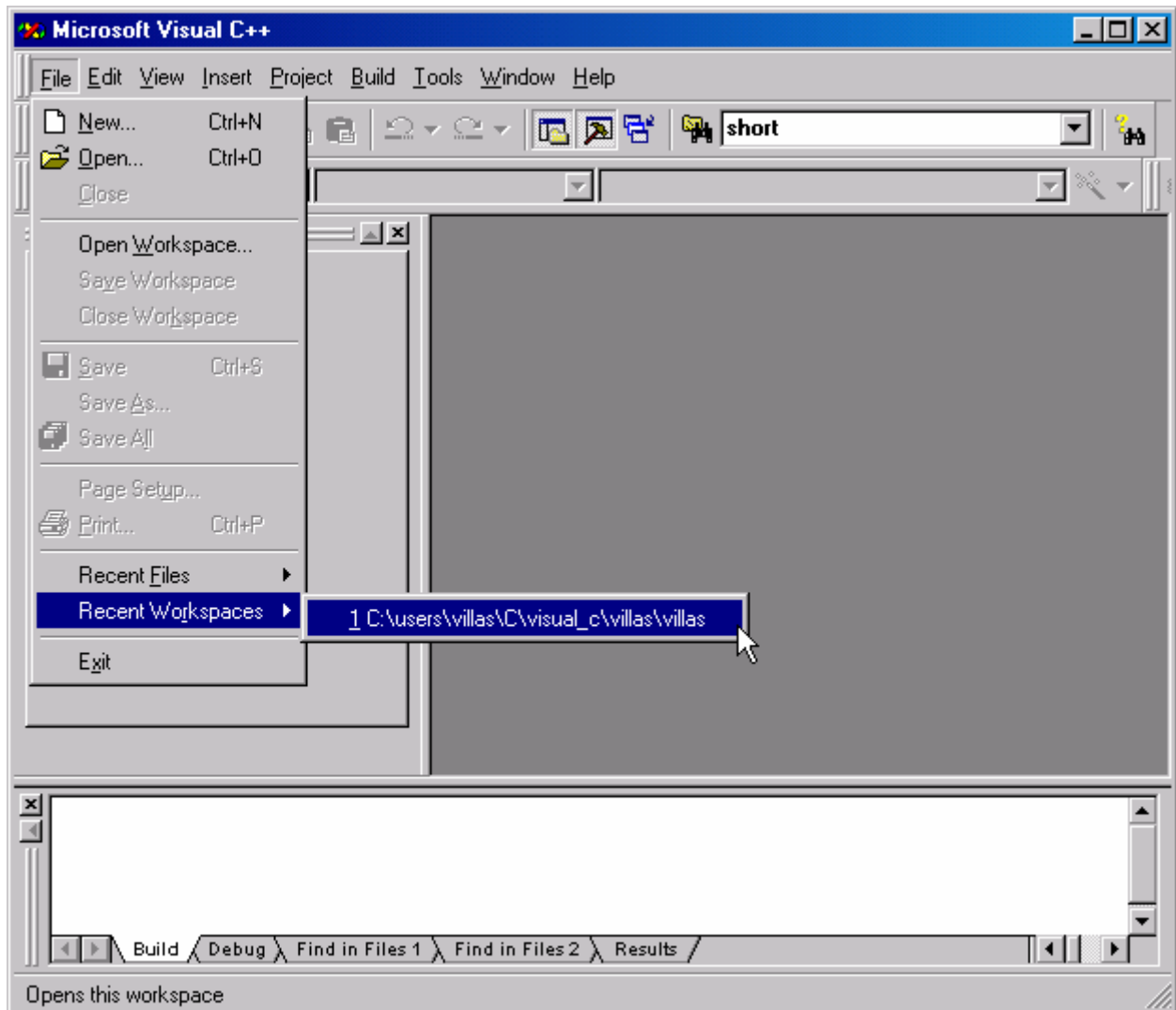
4.1.1 Reconhecendo o Compilador

O VisualC trabalha com o conceito de “workspace”. A idéia é que várias pessoas podem trabalhar com o mesmo compilador e o que uma pessoa faz não afeta o ambiente de trabalho de outra pessoa. A primeira coisa a fazer com esse compilador é criar um workspace para iniciar o trabalho. Para isso, vá no menu File - New. Na caixa de diálogo “New” que aparecer, selecione o tab “workspaces”. Digite um nome para o workspace no campo “Workspace name” (selecione também o local para o workspace). Um bom nome para o workspace é o seu nome pessoal (o nome pelo qual é conhecido, possivelmente o sobrenome;

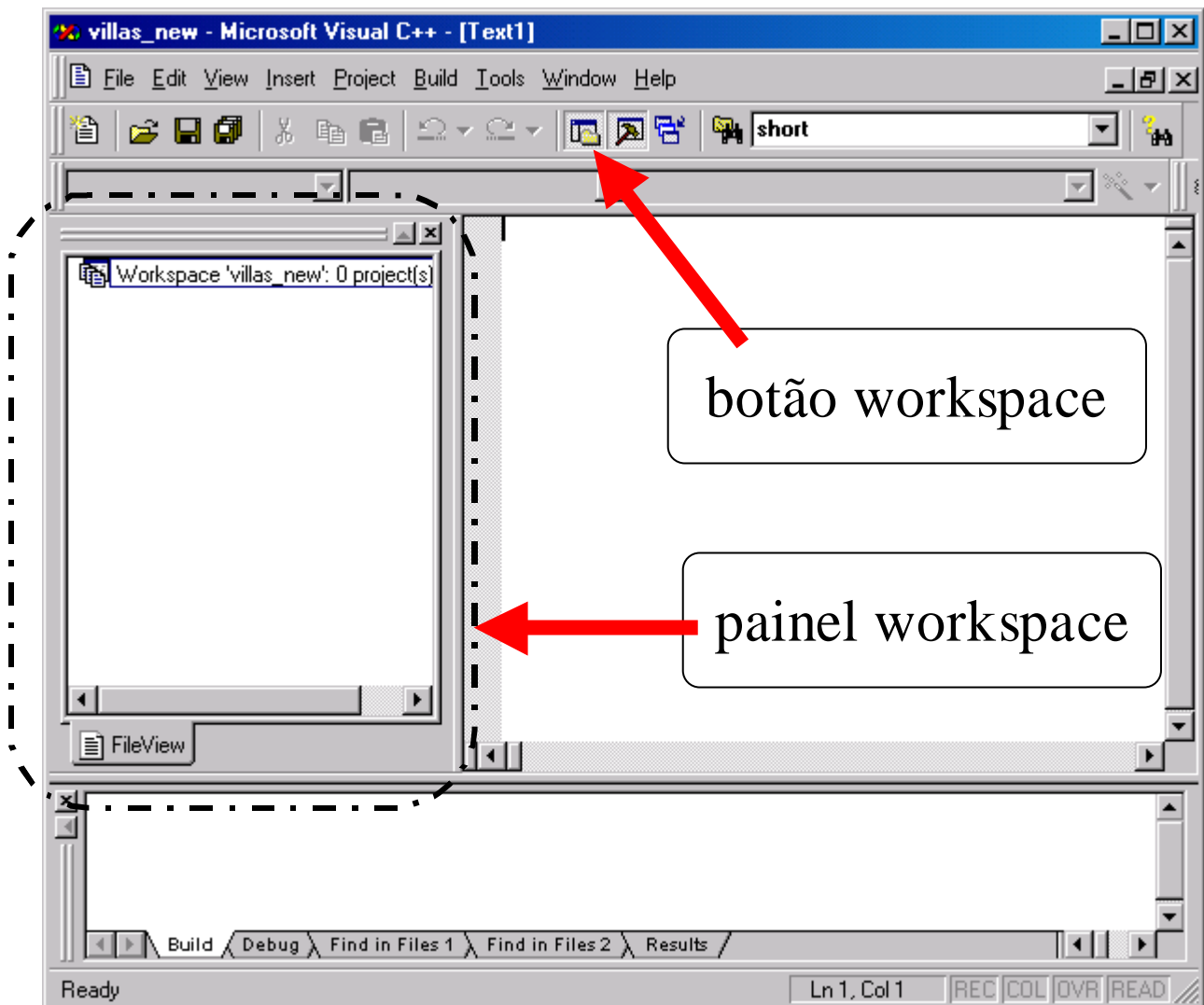
no meu caso uso “villas”; nesse exemplo, usei “villas_new”). Cria-se assim o arquivo “villas_new.dsw”



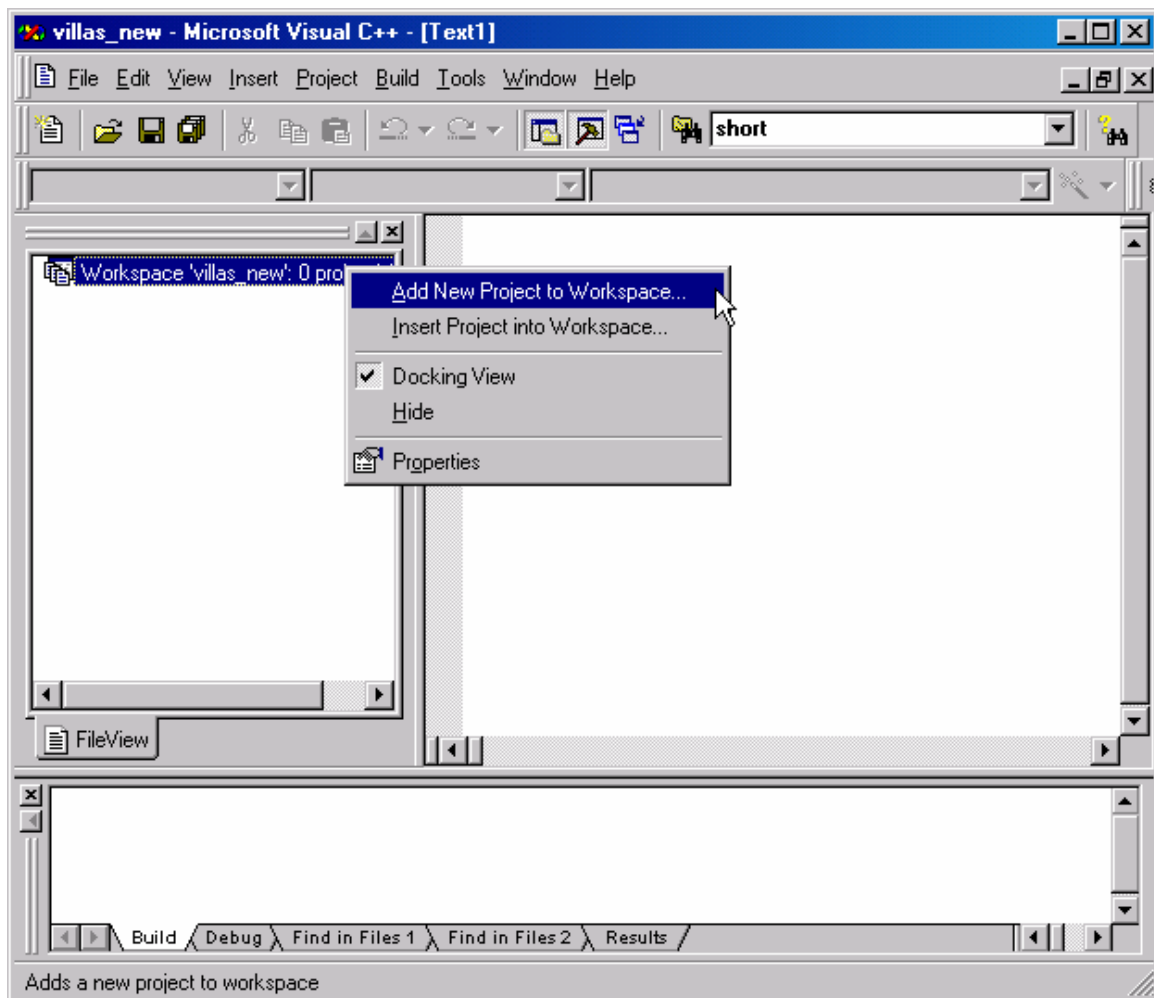
Ao entrar no VisualC uma próxima vez, abra o seu workspace e trabalhe dentro dele. Uma forma fácil de fazer isso é usar o comando pelo menu File - Recent Workspaces.



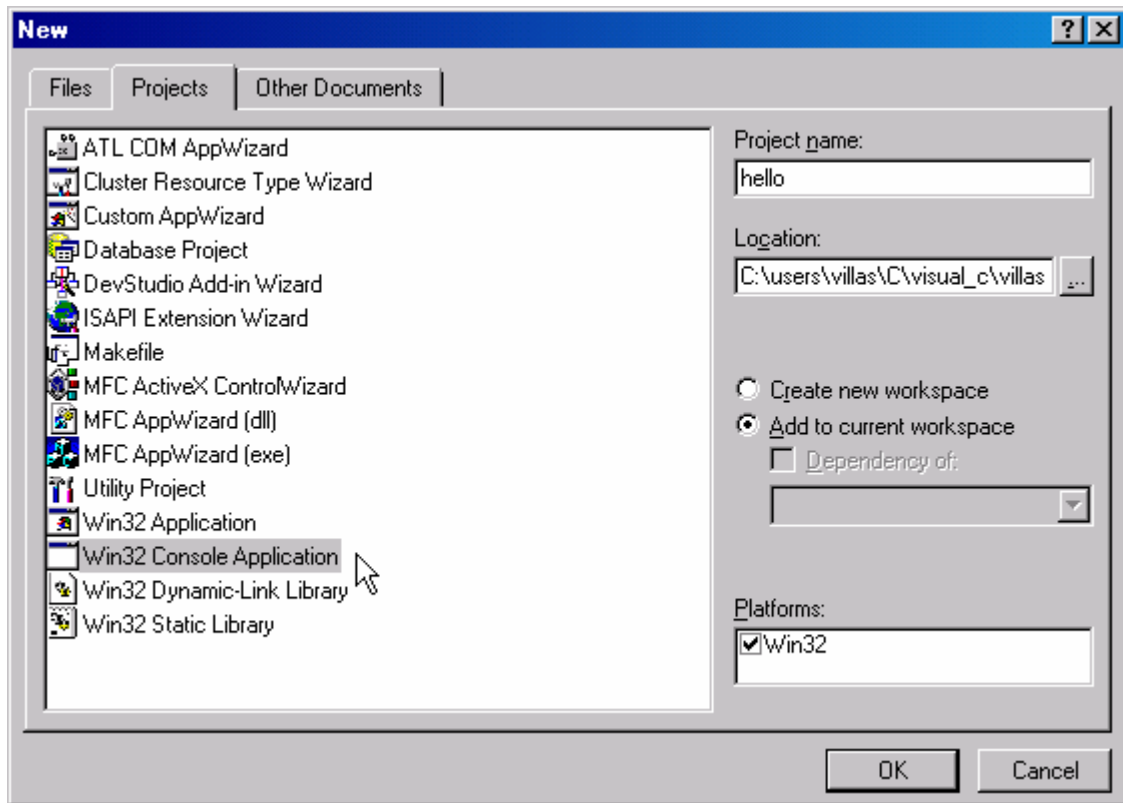
Um workspace pode conter diversos projetos. Cada projeto é uma aplicação, biblioteca, componente, etc., que se pretende desenvolver. Mantenha o “painel workspace” a mostra (use o botão “workspace” na barra de ferramentas para comutar (*toggle*) entre ver e não ver o painel workspace, que fica do lado esquerdo da tela).



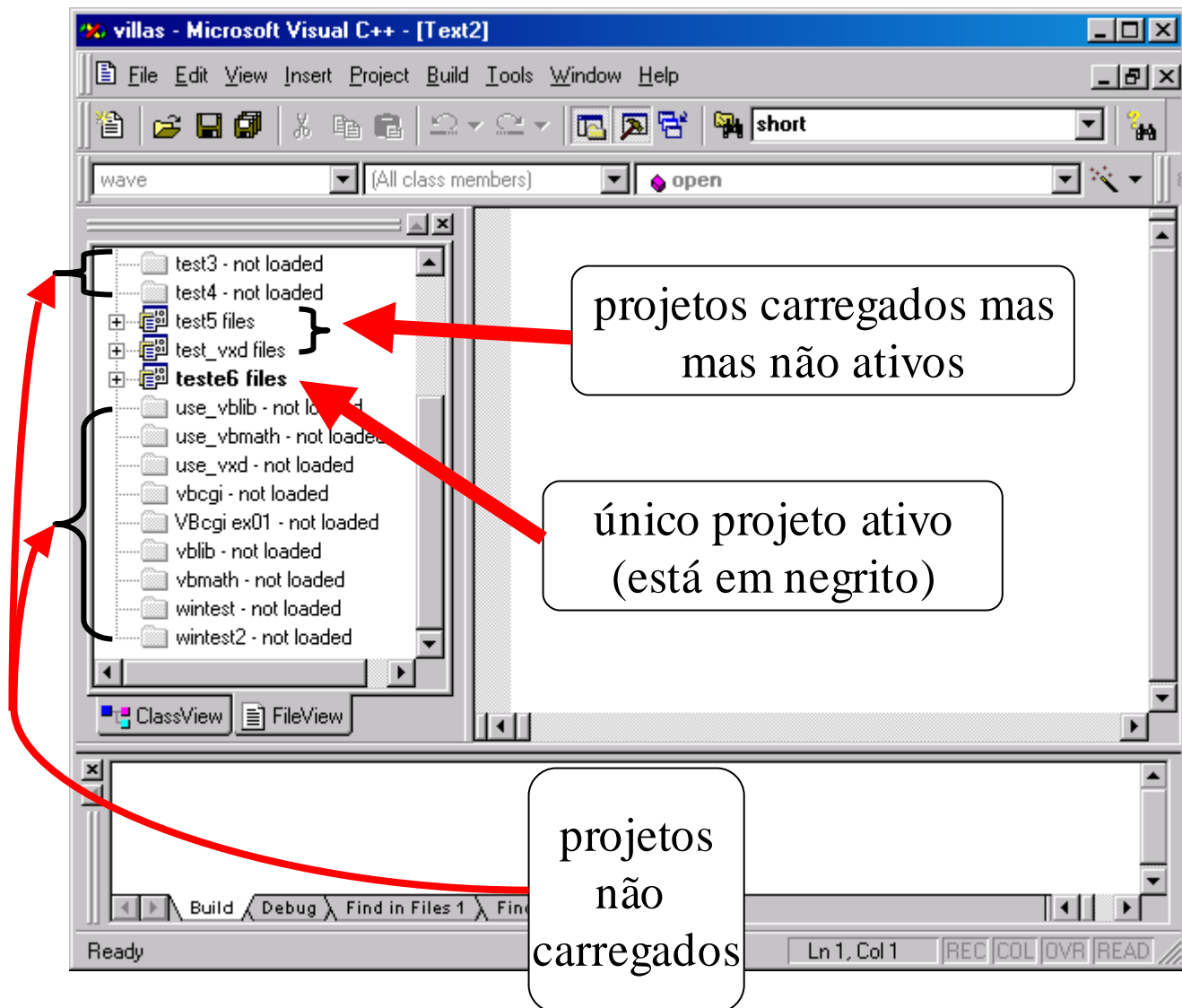
No painel workspace, clique com o botão direito sobre o ícone com nome “workspace ‘nome’ ”, e selecione “Add New Project to Workspace”.



No tab “projects” escolha o tipo de projeto que deseja fazer. Para iniciantes, escolha “win32 console application”. Dê um nome ao projeto (por exemplo “hello”) e clique OK.



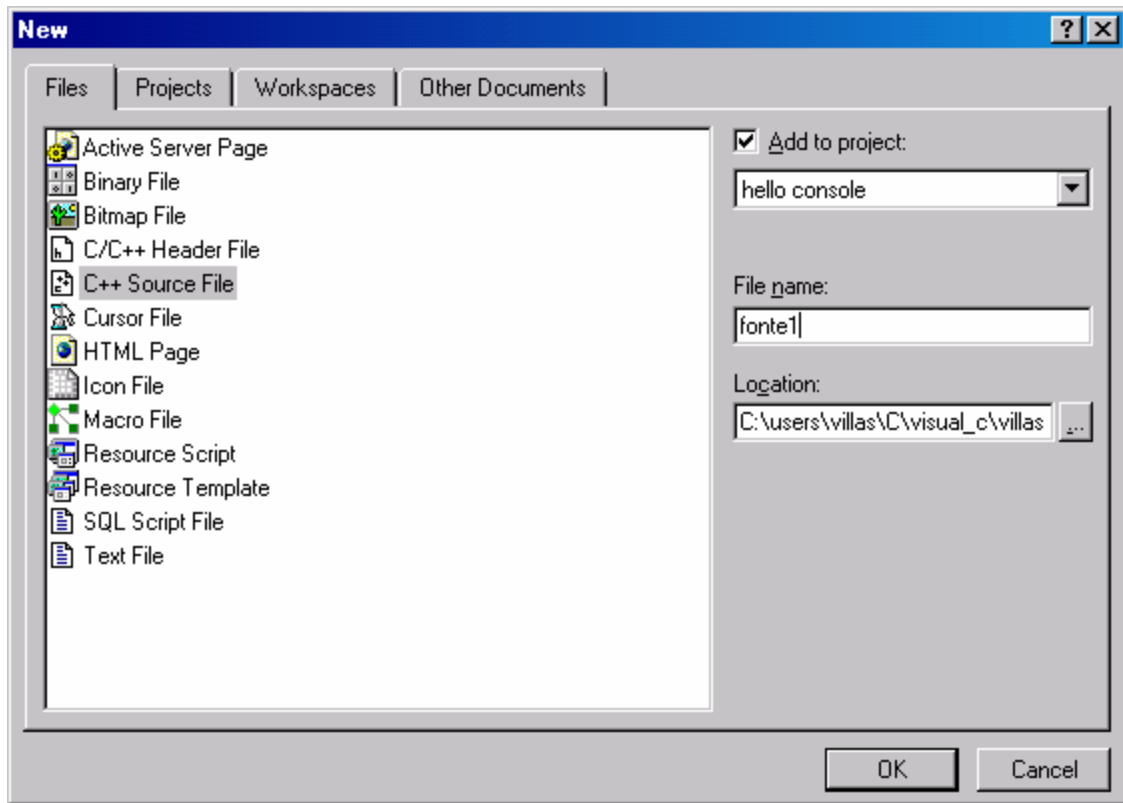
Dentro de um mesmo workspace, podem existir vários projetos. Dentre os projetos existentes, pode haver mais de um carregado (*loaded*). Dentre os projetos carregados, somente um está “active” (no foco). O único projeto ativo aparece com letras em negrito (*bold*), e é esse projeto que está sendo compilado e “debugado”.



4.1.2 “Hello world” para DOS

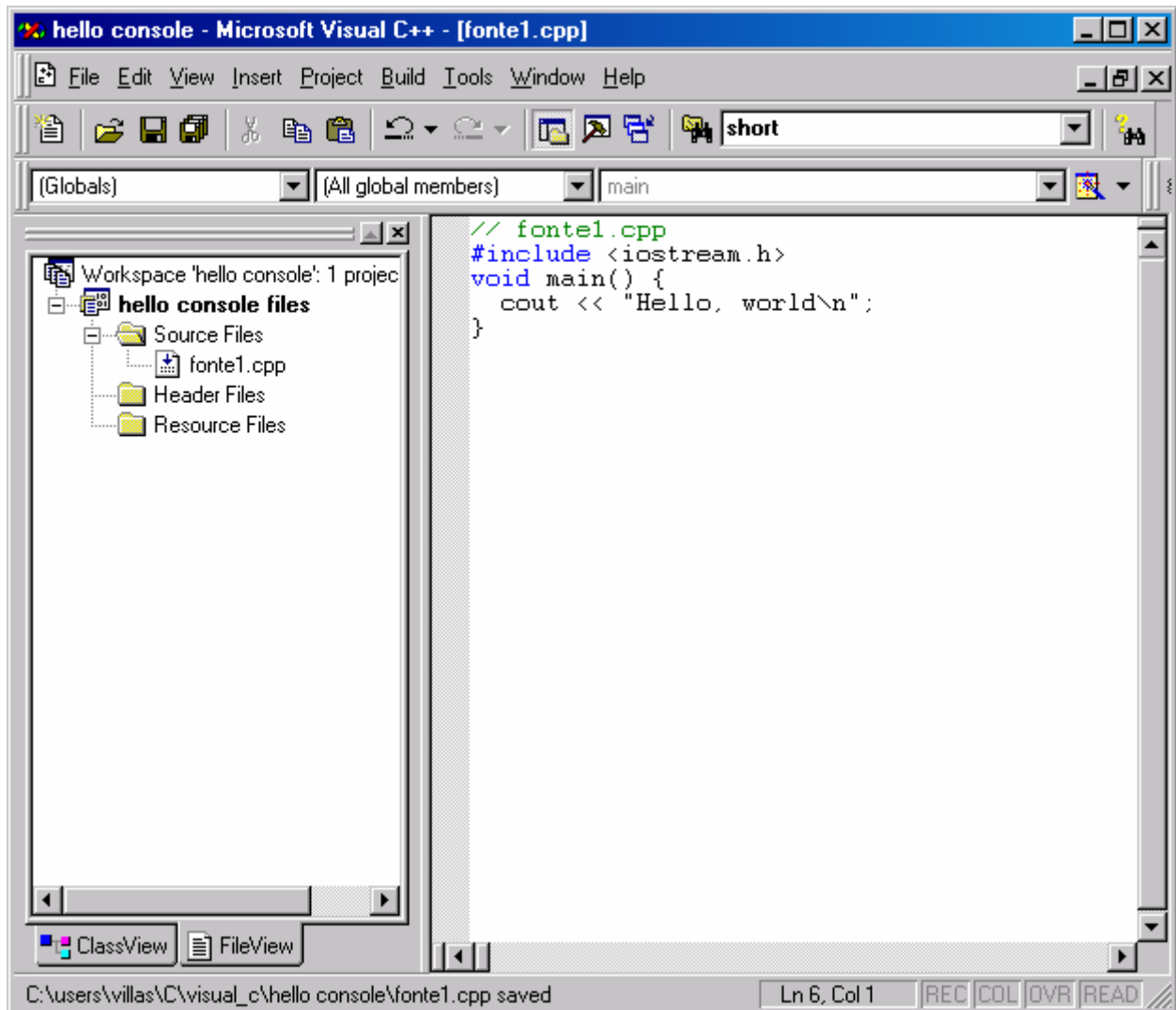
Crie um projeto com nome “hello console” do tipo “win32 console application”. No step 1 do application wizard, escolha “An empty project” (um projeto vazio, apesar da opção “Typical hello world application”). Clique “Finish”.

O seu projeto precisa de pelo menos um arquivo de código fonte. Crie-o com **File** - **New** - (no tab Files) <C++ source file> - no campo File name, escreva o nome do arquivo, digamos, “fonte1” (na verdade fonte1.cpp, mas a extensão .cpp é automática) - deixe ativo o flag “Add to project”, para que o seu arquivo fonte1.cpp seja automaticamente incluído no projeto em questão - <OK>.

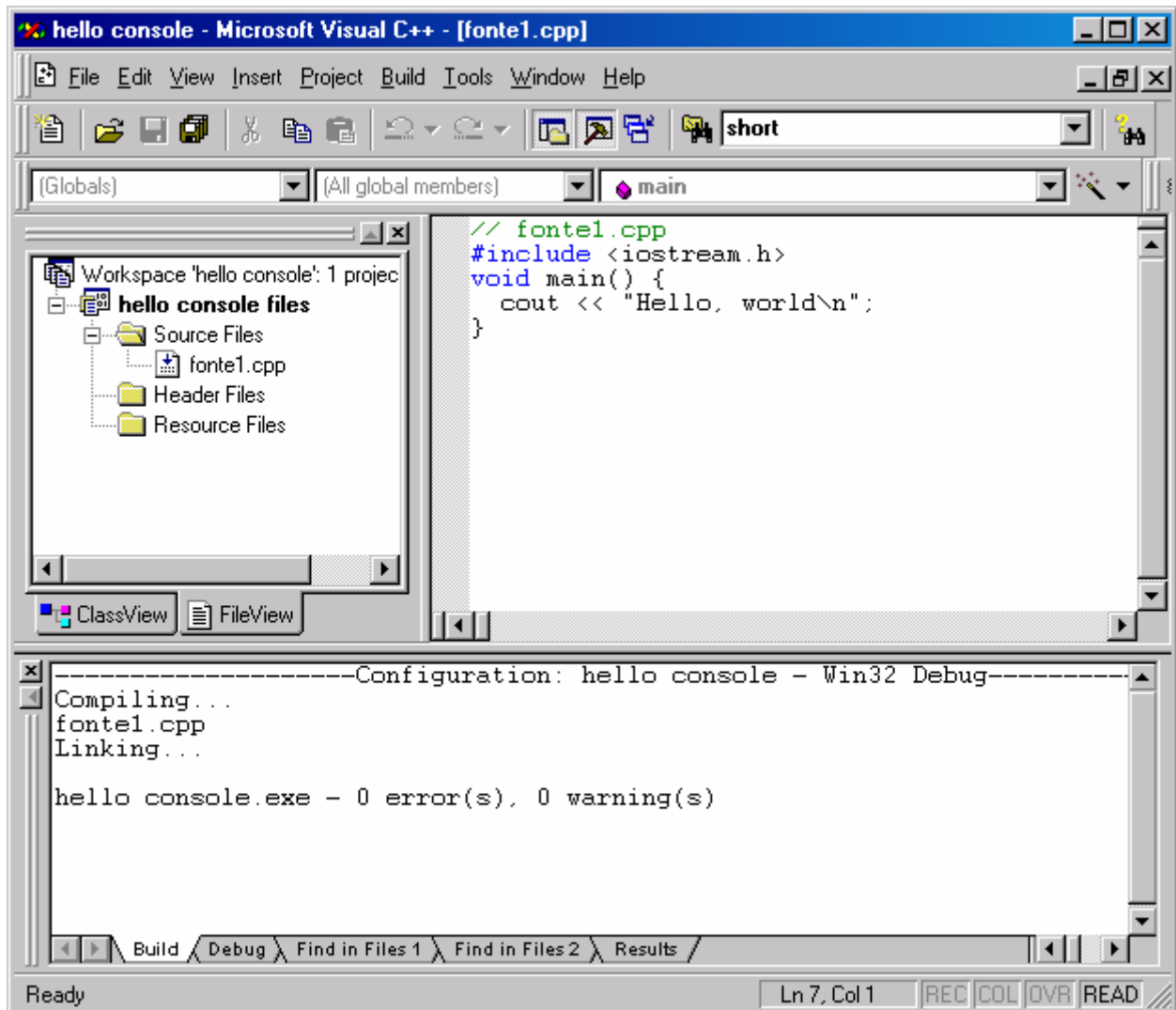


Confirme que a janela da direita está editando o arquivo fonte1.cpp olhando a barra de título do VisualC++, com “[fonte1.cpp]” do lado direito. Escreva o texto abaixo no arquivo que criado.

```
// fonte1.cpp
#include <iostream>
using namespace std;
int main() {
    cout << "Hello, world\n";
    return 0;
}
```



Compile e ligue (*link*) o programa com Build - Build (F7). Durante a compilação, na parte de baixo se abrirá uma janela mostrando as informações de compilação.



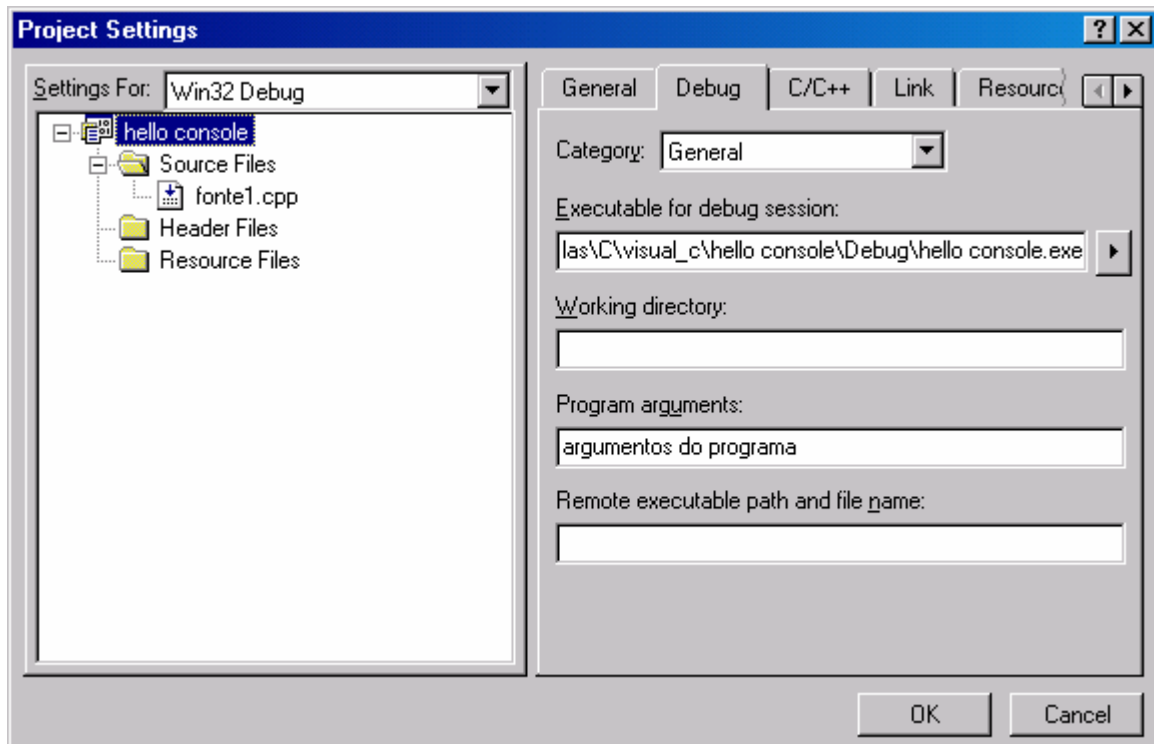
Pode-se rodar o programa sem debug, de dentro do VisualC++, com **Build - Execute** (<ctrl>F5). O mesmo programa também pode ser rodado “manualmente” a partir de uma janela DOS. Para isso, abra uma janela de DOS, vá para o diretório mostrado abaixo (ou o equivalente no seu sistema)

“C:\users\visual_c\ villas_new\hello console\Debug”

Lá há o programa “hello console.exe” que você acabou de compilar (no DOS aparece como “helloc~1.exe” devido ao problema de compatibilidade do DOS com arquivos com nomes maiores que 8 letras). Digite “helloc~1.exe” no prompt do DOS e veja o seu “hello world”.

4.1.2.1 Adicionando argumentos para a linha de comando

Comande **Projects - Settings** (<alt>F7) - <Debug tab> - no campo de <Program arguments>, digite os argumentos para o seu programa.



O programa abaixo lista os argumentos.

```
#include <iostream>
using namespace std;
int main(int argc, char**argv) {
    cout << "Hello, world\n";
    int i;
    for (i=0; i<argc; i++)
        cout << "Arg[" << i << "]: " << argv[i] << endl;
    return 0;
}
```

Resultado:

```
Hello, world
Arg[0]:C:\USERS\ villas_C\visual_c\ villas_new\hello console\Debug\hello console.exe
Arg[1]:argumentos
Arg[2]:do
Arg[3]:programa
```

4.1.3 Usando o Help

Aprender a usar bem o help é uma habilidade estratégica, pois é um conhecimento que permite ao programador usar o seu tempo para pesquisa pessoal no sentido de se aprimorar no uso da linguagem. Aprender a usar o help é “aprender a aprender”.

Como é fundamental usar o help do programa, isso torna-se mais um motivo para que nessa apostila se use o idioma inglês misturado com português. Para se encontrar algo no help, é preciso se ter em mente as palavras chave em inglês.

O help do Visual C++ 6.0 é muito grande, ocupando 2 CDs. Para não ocupar lugar demais no disco rígido do computador, a ferramenta de help vai copiando para o disco os pedaços do help a medida que vão sendo solicitados. Por isso, é sempre bom trabalhar com o Visual C++ tendo-se a mão os seus 2 CDs de help.

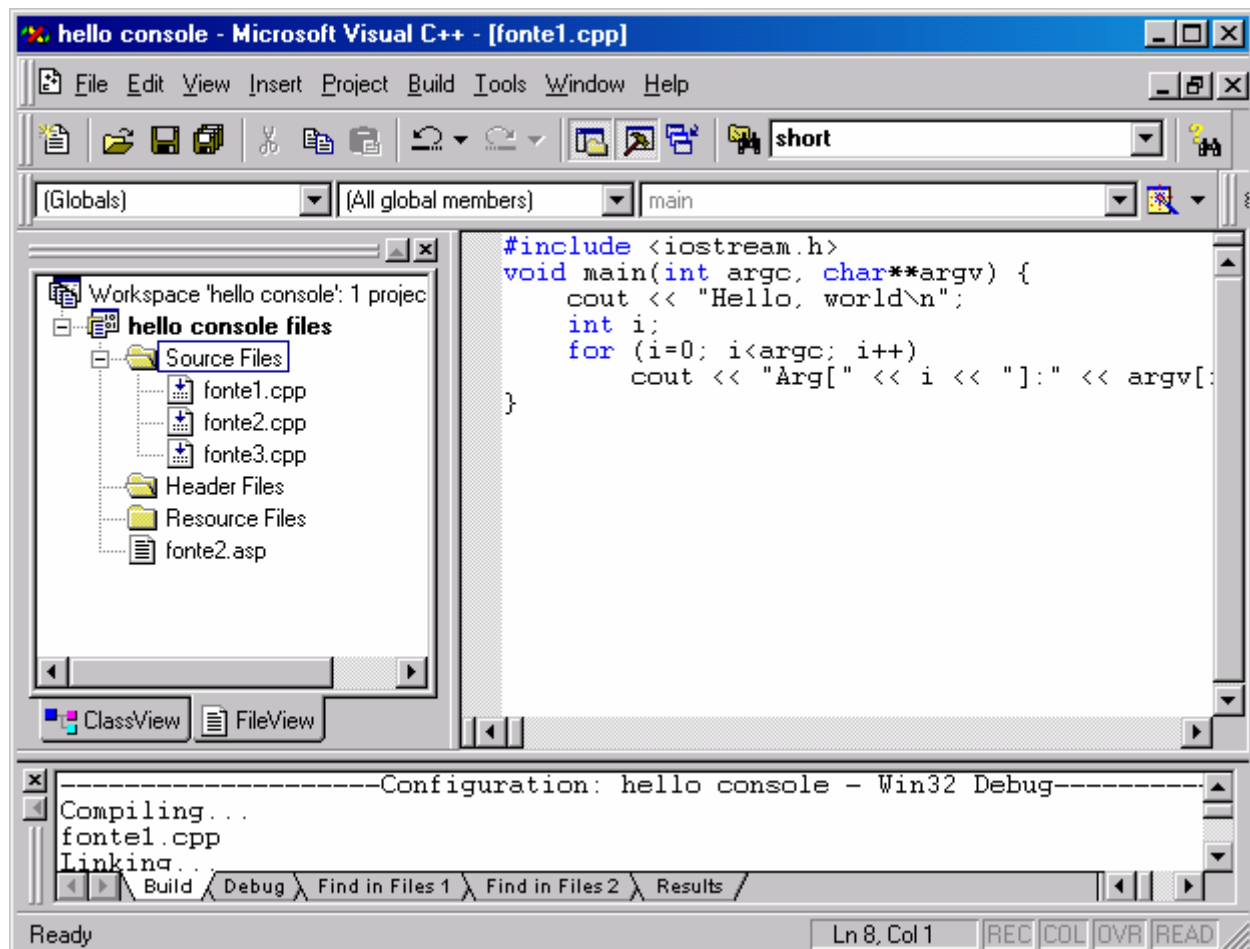
Uma forma direta de se usar o help é comandar Help - Search - e digitar o que se quer. Nessa apostila, em vários lugares recomenda-se que mais informação seja pesquisada no help , tópico “tal”, sendo que “tal” é uma palavra chave ou expressão chave faz o Visual C++ apontar para a parte correspondente do help.

4.1.4 Projetos (programas com múltiplos fontes)

Num “Workspace” do Visual C++, há diversos projetos. Cada projeto desses pode ter um ou mais arquivos fonte. Para compilar um programa que possui mais de um arquivo fonte, basta colocar todos os arquivos fonte que compõem o programa debaixo do mesmo folder “Source Files”.

Não é necessário acrescentar manualmente os arquivos header no folder “Header Files”. O Visual C++ faz isso automaticamente ao compilar os arquivos *.cpp.

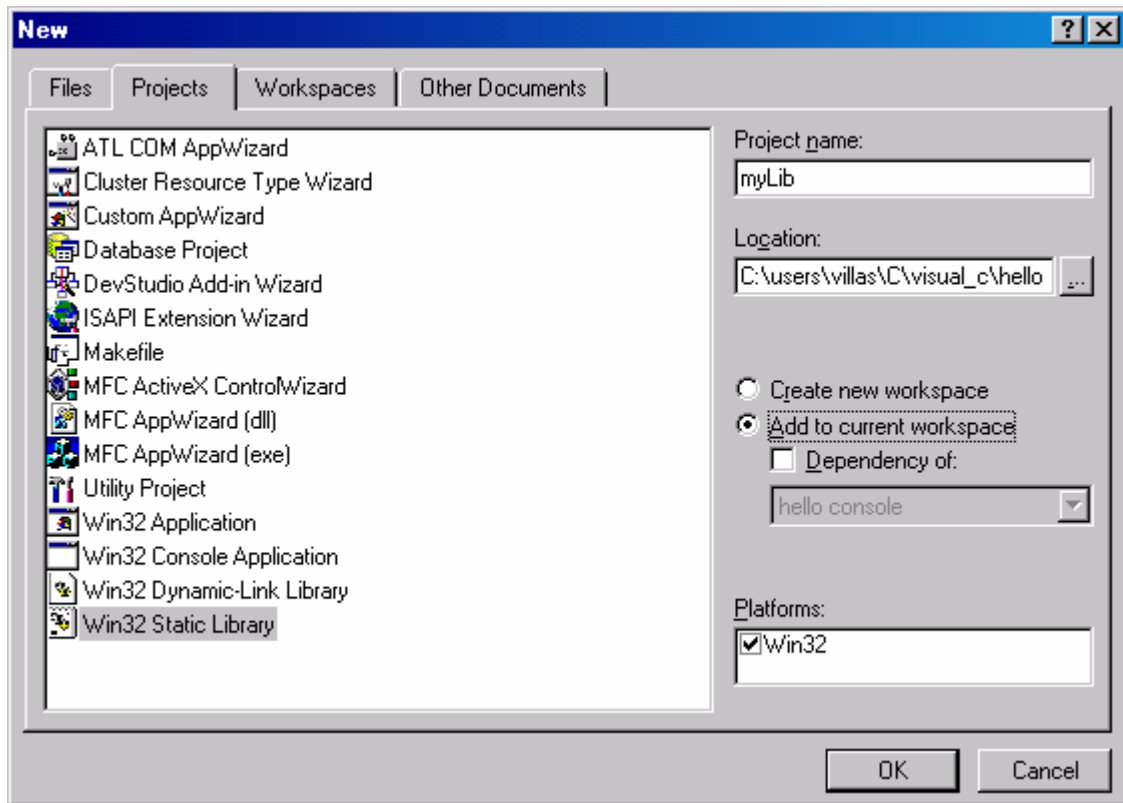
Além de arquivos *.cpp, pode-se também acrescentar no folder “Source Files” arquivos *.obj e *.lib, desde que compilados pelo Visual C++.



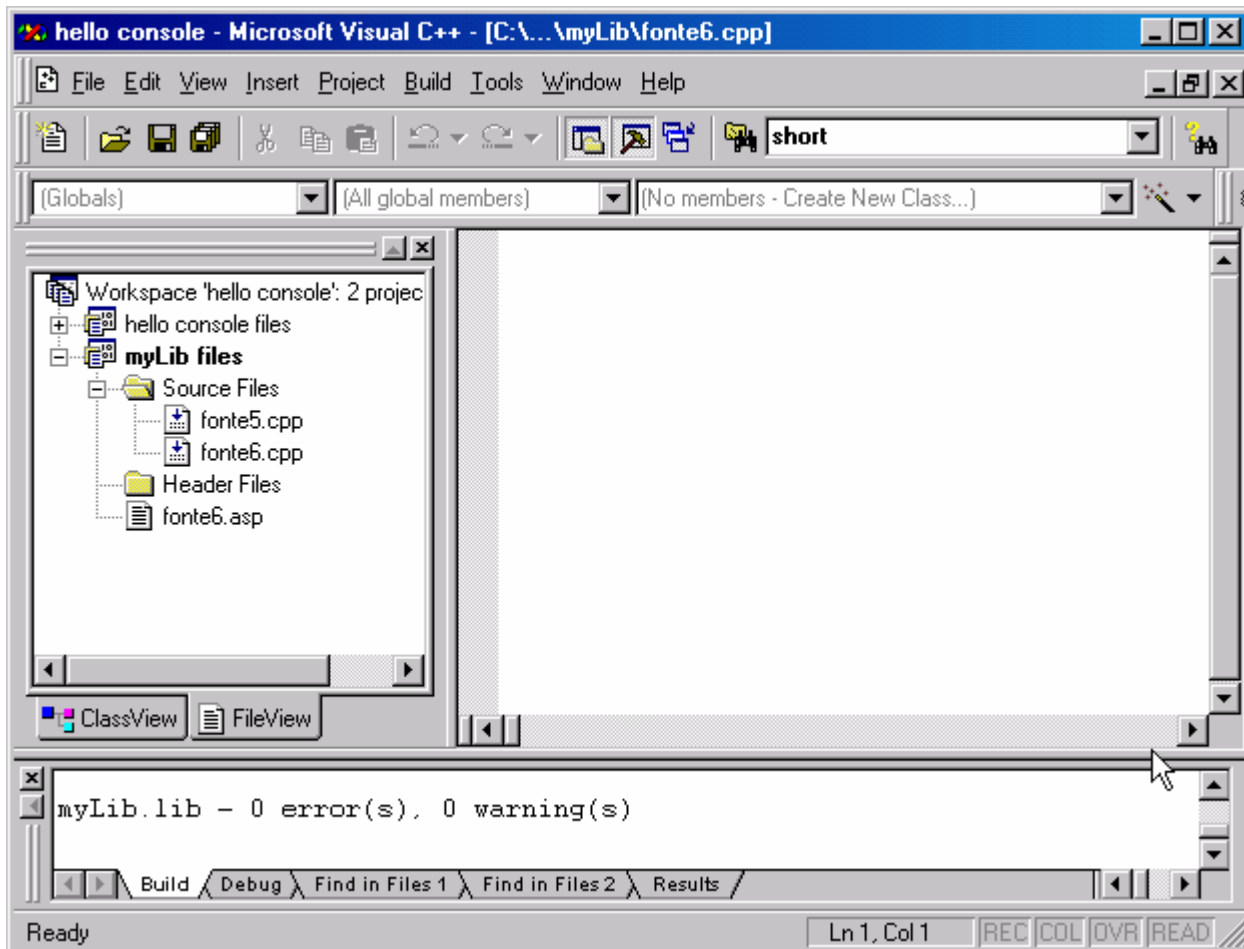
4.1.5 Bibliotecas

4.1.5.1 Fazer uma biblioteca

Para criar uma biblioteca estática (*.lib) com o Visual C++, comande **File - New**, e no tab “Projects” escolha “win 32 Static Library”. No exemplo da figura abaixo, foi escolhido o nome “myLib” para a biblioteca.



Acrescente arquivos fonte na biblioteca, como se o projeto fosse gerar um executável. Compile a biblioteca da mesma forma como se compila um executável (F7). Também da mesma forma como num programa executável, escolha a versão (debug, release) da biblioteca (veja página 99).



4.1.5.2 Incluir uma biblioteca num projeto

Criar uma biblioteca é um projeto, mas a lib gerada não é executável. Para incluir uma biblioteca num outro projeto que gera um programa executável, basta acrescentar a biblioteca no folder “Source Files” do projeto que gera executável e mandar compilar.

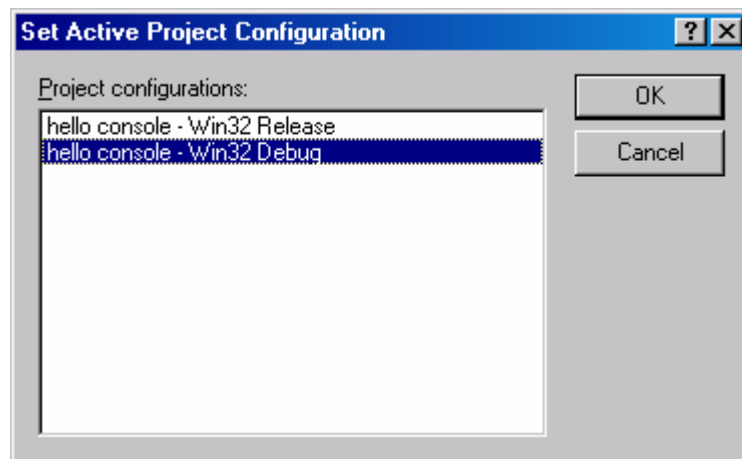
4.1.5.3 Examinar uma biblioteca

// todo-todo

4.1.6 Debug

Debugar um programa significa executa-lo passo a passo, observando as variáveis e conferindo se o comportamento encontra-se conforme esperado. Para que um programa possa ser debugado, é preciso que seja compilado para essa finalidade, isto é, que o compilador salve no arquivo executável final informações para possibilitar o debug. Obviamente essa informação de debug ocupa espaço, e esse espaço é desnecessário na versão final (*release version*) do programa. Portanto, quando se compila um programa é preciso saber se o compilador está

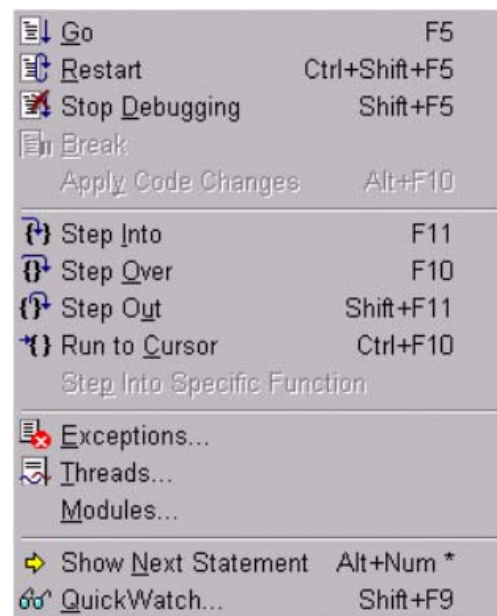
configurado para compilar a debug version ou a release version. No Visual C++, isso é feito com **Build - Set Active Configuration**, e selecionando a versão desejada. Para um mesmo programa, compile-o na versão release e debug e compare os tamanhos do arquivo executável gerado. Para um programa “hello world”, o tamanho da versão debug é 173K, enquanto o tamanho da versão release é 36K.



Quando se inicia o debug de um programa, surge o menu de Debug. As opções de debug desse menu estão mostradas abaixo.

Resumo dos Comandos do Debug

Go	F5
Restart	Ctrl+Shift+F5
Stop Debugging	Shift+F5
Step Into	F11
Step Over	F10
Step Out	Shift+F11
Run to Cursor	Ctrl+F10
Quick Watch	Shift+F9
Toggle Break Point F9	



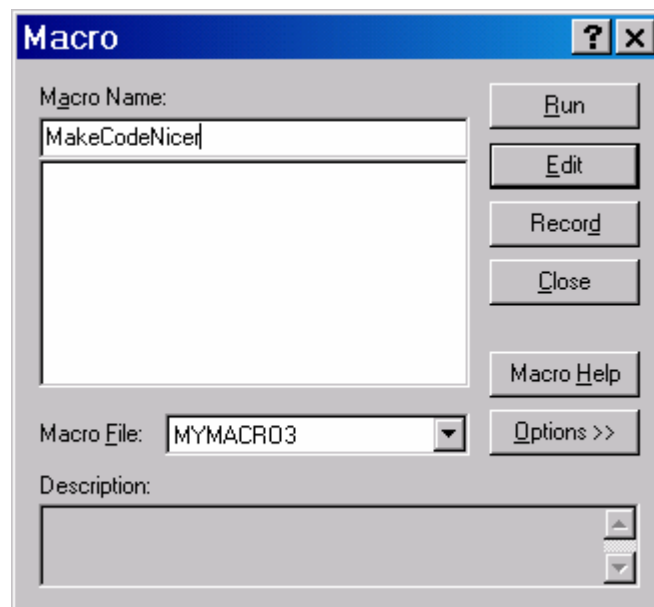
A única forma de compreender o funcionamento do debug é praticando-o. Portanto, pratique os comandos de debug acima até que se sintam a vontade com a sua utilização.

4.1.7 Dicas extras

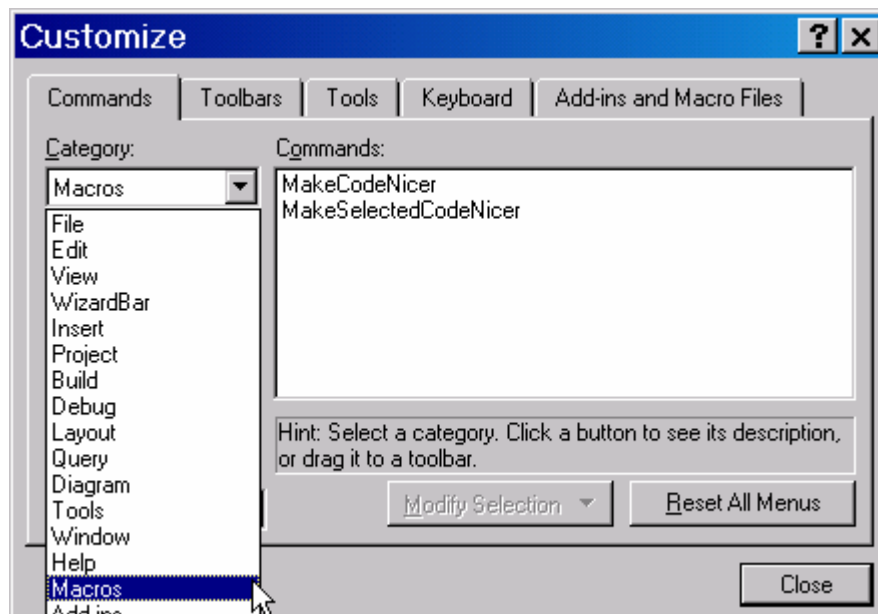
4.1.7.1 Macro para formatação do texto fonte

Uma opção muito útil para apoiar a programação é utilizar uma macro para apoiar a formatação do texto. A macro abaixo, chamada de MakeCodeNicer (faça o código ficar mais simpático), desenvolvida originalmente por Alvaro Mendez, é escrita baseado na função “SmartFormat” da Microsoft. Essa macro implementa a reformatação.

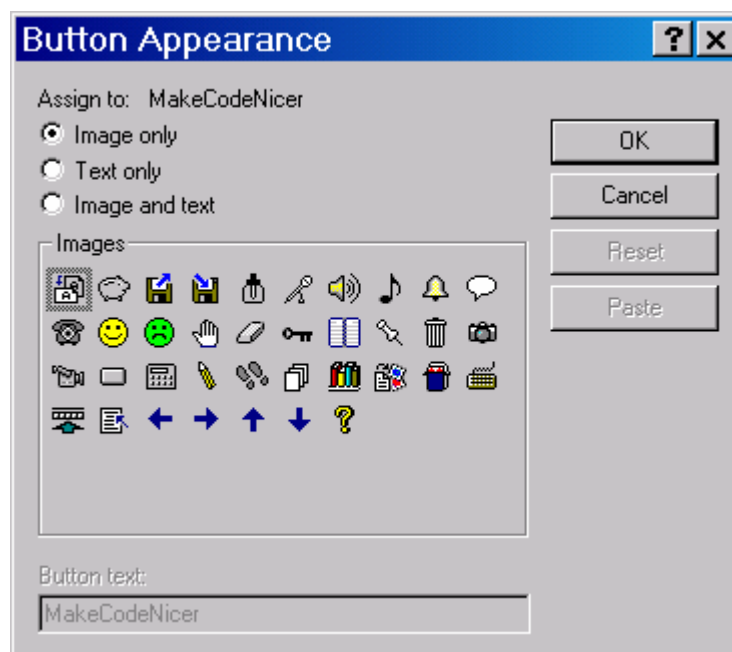
Para acrescentar essa macro ao seu Visual C++, faça o seguinte. Em primeiro lugar, corte e cole o texto da macro para um arquivo de texto. Não digite o conteúdo, pois seria muito trabalhoso. No menu do Visual C++, selecione Tools - Macro, digite MakeCodeNicer e selecione o botão Edit. Na janela de descrição que aparecer, coloque o que quiser (pode-se não colocar nada).



Corte e cole o conteúdo da macro para o editor de texto do Visual C++. Salve o trabalho (File-Save), e pode fecha-lo. Para facilitar o uso da macro, pode-se criar um atalho para que seja chamada. Faça isso por comandar pelo menu Tools - Customize, e escolha Macros na combo box de Category.



Em seguida, clique em MakeCodeNicer, e arraste até a barra de ferramentas, e solte o mouse no local que gostaria de ter um botão para chamar a macro. Deverá surgir uma caixa de diálogo “Button Appearance” (aparência do botão), como mostrado abaixo. Escolha qualquer um, digamos a cara alegre (amarela).



Com isso, deverá aparecer o botão na barra de ferramentas do Visual C, como mostrado na figura abaixo.



Testando a macro. Seja o código abaixo, claramente mal indentado.

```
#include "VBMcgi.h"
int main()
{ VBMcgi cgi;  cgi.htmlCompleteHeader();  cout << "<html><body>" << endl;
  cout << "Some text in html" << endl;    cout << "</body></html>" << endl; return 0;}
```

Após a reformatação, o código ficará como abaixo.

```
#include "VBMcgi.h"
int main()
{
    VBMcgi cgi;
    cgi.htmlCompleteHeader();
    cout << "<html><body>" << endl;
    cout << "Some text in html" << endl;
    cout << "</body></html>" << endl;
}
```

O código da macro MakeCodeNicer é mostrado abaixo (corte-o e cole-o para seu editor de macro).

```
'-----
'FILE DESCRIPTION: Routines to reformat C/C++ code.
'http://www.del.ufrj.br/~villas/livro_c++.html
'-----

Sub MakeCodeNicer()
'DESCRIPTION: Reformats the source code to look nicer, the way I like it.
' Written by Alvaro Mendez on 06/10/1999
' Last Updated on 02/10/2000

' Check that the current document can be changed
if ActiveDocument.ReadOnly then

    ' If we're connected to SourceSafe, let it prompt for check out
    ActiveDocument.Selection = " "
    ActiveDocument.Undo

    if ActiveDocument.ReadOnly then          ' check again
        MsgBox "This macro cannot be executed on a read-only file.", _
            vbExclamation
        exit sub
    end if
end if

' Save current line so we can return to it at the end
dim nLine
nLine = ActiveDocument.Selection.CurrentLine

' Add spaces in a few places and get rid of it in others
Replace "\:b+;", ";"
Replace "\:b+::\:b+", ":@"
Replace "\:b+", "("
Replace "=(", "= ("
Replace "if(", "if ("
Replace "for(", "for ("
Replace "while(", "while ("
Replace "switch(", "switch ("
Replace "catch(", "catch ("
Replace "return(", "return ("
Replace "(\:b+", "("
Replace "\:b+)", ")"
```

```

Replace ";)", "; )"
Replace ";;\b+)", ";;)"
Replace "\[\b+", "["
Replace "\b+]", "]"
Replace "\b+\[", "["

' Make sure these statements don't end on the same line they started.
BreakSingleLinners "if ("
BreakSingleLinners "for ("
BreakSingleLinners "switch ("

' Break up any lines containing multiple statements
BreakLinesWithMultipleStatements

' Make sure braces are on lines by themselves (unless followed by comments)
IsolateOnLeft "{"
IsolateOnRight "{"
IsolateOnRight "}"
IsolateOnLeft "}"

' Break up simple else statements on the same line (except "else if")
Replace "else\b+if (", "elseif("
IsolateOnRight "else\b+"
Replace "elseif(", "else if ("

' Break up case statements appearing on single lines
IsolateOnRight "case .+:\b+"
IsolateOnRight "default:\b+"
IsolateOnLeft "break;"

' Add a space between these operators
FixOperator "=", 1
FixOperator "==", 2
FixOperator "!=", 2
FixOperator "\+=", 2
FixOperator "-=", 2
FixOperator "\*=", 2
FixOperator "/=", 2
FixOperator "\+", 1
FixOperator "-", 1
FixOperator "<=", 2
FixOperator ">=", 2
FixOperator "<<", 2
FixOperator ">>", 2
FixOperator "&&", 2
FixOperator "||", 2
FixOperator "|", 1
FixOperator "?", 1
FixLessThanOperator
FixExponents

' Append a space after these
AppendSpace ","
AppendSpace ";"

' Make sure the first C++ comment of every line has a space after it.
InsertSpaceAfterFirstInLineComment

' Replace all the trailing whitespace (thanks to Paul Bludov)
ActiveDocument.Selection.ReplaceText "\b+(\$\*)", "\1", dsMatchRegExp

' Fix tabs within code surrounded by braces

```

```

TabifyMatchingBraces

' Remove any lines that are considered extraneous (ie. extra blank lines)
RemoveExtraneousLines

' Indent every "case" inside switch statements (thanks to Jim Cooper)
IndentSwitchBody

' Go back to where we were at the beginning
ActiveDocument.Selection.GoToLine nLine

End Sub

' Is the cursor currently within a quoted string (or character)
function IsWithinQuotes
    dim nCurrentLine, nCurrentColumn, iPos, strBuffer, nCount

    nCurrentLine = ActiveDocument.Selection.CurrentLine
    nCurrentColumn = ActiveDocument.Selection.CurrentColumn

    ActiveDocument.Selection.Cancel
    ActiveDocument.Selection.StartOfLine dsFirstText, dsExtend

    nCount = 0
    iPos = 0
    strBuffer = ActiveDocument.Selection

    ' Count all occurrences of a double quote which apply to quoted strings
    do while true
        iPos = InStr(iPos + 1, strBuffer, "\"", vbTextCompare)
        if not (iPos > 0) then
            exit do
        end if

        if iPos = 1 then ' if it's the first character, then it's valid
            nCount = nCount + 1
        else
            ' Make sure it's not preceded by a \ or a \\
            if Mid(strBuffer, iPos - 1, 1) <> "\" then
                nCount = nCount + 1
            elseif (iPos > 2) and (Mid(strBuffer, iPos - 2, 1) = "\"") then
                nCount = nCount + 1
            end if
        end if
    loop

    ' If number of quotes is odd, we must be inside a quoted string!
    IsWithinQuotes = ((nCount > 0) and ((nCount Mod 2) <> 0))

    ActiveDocument.Selection.MoveTo nCurrentLine, nCurrentColumn

    ' If we're not inside a quoted string, check for a quoted character
    if not IsWithinQuotes then
        ActiveDocument.Selection.CharLeft dsExtend

        ' If we find a quoted character left of us, check for one on the right
        if ActiveDocument.Selection = "\"" then
            ActiveDocument.Selection.CharRight
            ActiveDocument.Selection.CharRight dsExtend
        if ActiveDocument.Selection = "\" then
            ActiveDocument.Selection.CharRight
            ActiveDocument.Selection.CharRight dsExtend
        end if
    end if
end function

```



```

        end if
        ActiveDocument.Selection.CharRight
        ActiveDocument.Selection.CharRight dsExtend

        if ActiveDocument.Selection = "" then
            IsWithinQuotes = true
        end if
    end if
    ActiveDocument.Selection.MoveTo nCurrentLine, nCurrentColumn
end if

' If we're inside quotes, proceed from the next character
if IsWithinQuotes then
    ActiveDocument.Selection.CharRight
end if

end function

' Is current selection preceded by a C++ comment? ("//")
function IsWithinComment
    dim nCurrentLine, nCurrentColumn

    nCurrentLine = ActiveDocument.Selection.CurrentLine
    nCurrentColumn = ActiveDocument.Selection.CurrentColumn

    ActiveDocument.Selection.Cancel
    ActiveDocument.Selection.StartOfLine dsFirstText, dsExtend

    IsWithinComment = false
    if (InStr(1, ActiveDocument.Selection, "//", vbTextCompare) > 0) then
        IsWithinComment = true        ' since it's commented out
        nCurrentLine = nCurrentLine + 1 ' we proceed from the next line
    end if

    ActiveDocument.Selection.MoveTo nCurrentLine, 1

end function

' Is the cursor on a line with a preprocessor macro? ("#define ")
function IsPreprocessorMacro
    dim nCurrentLine, nCurrentColumn

    nCurrentLine = ActiveDocument.Selection.CurrentLine
    nCurrentColumn = ActiveDocument.Selection.CurrentColumn

    IsPreprocessorMacro = false
    ActiveDocument.Selection.EndOfLine dsExtend

    if Mid(LTrimTabs(ActiveDocument.Selection), 1, 8) = "#define " then
        IsPreprocessorMacro = true        ' since it's in a macro
        nCurrentLine = nCurrentLine + 1    ' we proceed from the next line
    end if

    ActiveDocument.Selection.MoveTo nCurrentLine, nCurrentColumn

end function

' Should the current selection be ignored?
' (ie., is it within a comment, between quotes, or inside a macro?)
function ShouldIgnore

    ShouldIgnore = false

```

```

    if IsWithinQuotes then
        ShouldIgnore = true
        exit function
    end if

    if IsWithinComment then
        ShouldIgnore = true
        exit function
    end if

    if IsPreprocessorMacro then
        ShouldIgnore = true
    end if

end function

' Put the cursor at the top of the document and return "" to be passed
' initially to GetCurrentPosition
function InitializePosition
    ActiveDocument.Selection.StartOfDocument
    InitializePosition = ""
end function

' Retrieve the current position and return true if it's greater than the
' last one. This is used to ensure that the file is only searched once
' (provided the search is started from the top)
function GetCurrentPosition(strPos)
    dim nLastLine, nLastColumn, nCurrentLine, nCurrentColumn, iPos, ch

    nLastLine = -1
    nLastColumn = -1

    nCurrentLine = ActiveDocument.Selection.CurrentLine
    nCurrentColumn = ActiveDocument.Selection.CurrentColumn

    ' Parse the last position and extract the line and column
    for iPos = 1 to Len(strPos)
        ch = Mid(strPos, iPos, 1)
        if ch = "," then
            nLastLine = Int(Mid(strPos, 1, iPos))
            nLastColumn = Int(Mid(strPos, iPos + 1))
            exit for
        end if
    next

    ' Return true if we're currently past the last position
    strPos = nCurrentLine & "," & nCurrentColumn
    GetCurrentPosition = (nCurrentLine > nLastLine) or _
        ((nLastLine = nCurrentLine) and (nCurrentColumn > nLastColumn))

end function

' Move by a certain number of columns
sub MoveByColumns(nBy)
    ActiveDocument.Selection.MoveTo ActiveDocument.Selection.CurrentLine, _
        ActiveDocument.Selection.CurrentColumn + nBy
end sub

' Replace the given strFrom with strTo case sensitively
sub Replace(strFrom, strTo)
    dim strLastPos, bContinue

```

```

strLastPos = InitializePosition
do while ActiveDocument.Selection.FindText(strFrom, _
    dsMatchCase + dsMatchRegExp)

    bContinue = GetCurrentPosition(strLastPos)

    ' Check if we're inside a comment or between quotes
    if not ShouldIgnore then

        ' Repeat the search since ShouldIgnore puts the cursor at
        ' the beginning of the line
        ActiveDocument.Selection.FindText strFrom, _
            dsMatchCase + dsMatchRegExp
        ActiveDocument.Selection = strTo

    elseif not bContinue then
        exit do
    end if
loop
end sub

' Break the given str ending in (, so that instead of this:
'   if (a) return b;
' it looks like this:
'   if (a)
'       return b;
sub BreakSingleLiners(str)
    dim strLastPos, strFound, nCol, bBreak, strAfterFound

    ' Verify str ends in (, the beginning parenthesis
    if Right(str, 1) <> "(" then
        exit sub
    end if

    strLastPos = InitializePosition

    while ActiveDocument.Selection.FindText(str, dsMatchCase) and _
        GetCurrentPosition(strLastPos)

        ' Check if we're inside a comment or between quotes
        if not ShouldIgnore then

            ' Repeat the search since ShouldIgnore puts the cursor at the
            ' beginning of the line
            ActiveDocument.Selection.FindText str, dsMatchCase

            ' Find the matching brace and go to the end of the line
            ActiveDocument.Selection.CharRight
            ActiveDocument.Selection.CharLeft
            ExecuteCommand "GoToMatchBrace"
            ActiveDocument.Selection.CharRight
            nCol = ActiveDocument.Selection.CurrentColumn
            ActiveDocument.Selection.EndOfLine dsExtend
            strFound = LTrimTabs(ActiveDocument.Selection)

            ' If there's anything after the brace that isn't a comment, move
            ' it to the next line
            if (Len(strFound) > 0) and (Left(strFound, 1) <> "/" ) then
                bBreak = false
            end if
        end if
    end while
    strAfterFound = strFound
end sub

```

```

        ' Check if there's a "{" after the statement which should
        ' also be broken into multiple lines
        if (Mid(strFound, 1, 1) = "{") and (Len(strFound) > 1) then
            strAfterFound = LTrimTabs(Mid(strFound, 2))
            if strAfterFound <> "" then
                ActiveDocument.Selection = "{" + strAfterFound
                ActiveDocument.Selection.MoveTo _
                    ActiveDocument.Selection.CurrentLine, nCol
                ActiveDocument.Selection.NewLine
                ActiveDocument.Selection.CharRight
                ActiveDocument.Selection.NewLine
                bBreak = true ' primitive but it works
            end if
        end if

        if not bBreak then
            ActiveDocument.Selection = strFound
            ActiveDocument.Selection.MoveTo _
                ActiveDocument.Selection.CurrentLine, nCol
            ActiveDocument.Selection.NewLine
        end if
    end if
end if
wend

end sub

' Trim blanks AND tabs from the left side
function LTrimTabs(str)
    dim iPos, ch

    for iPos = 1 to Len(str)
        ch = Mid(str, iPos, 1)
        if ch <> " " and ch <> vbTab then
            exit for
        end if
    next

    LTrimTabs = Mid(str, iPos)
end function

' Isolate the given str on a new line with nothing left of it
sub IsolateOnLeft(str)
    dim strLastPos, nLen, bContinue, nCurrentLine, nCurrentColumn

    strLastPos = InitializePosition

    while ActiveDocument.Selection.FindText("^.*" & str, dsMatchRegExp + _
        dsMatchCase) and GetCurrentPosition(strLastPos)

        ' Check if we're inside a comment or between quotes
        if not ShouldIgnore then

            ' Repeat the search since ShouldIgnore puts the cursor at the
            ' beginning of the line
            ActiveDocument.Selection.FindText "^.*" & str, _
                dsMatchRegExp + dsMatchCase

            ' Get the length of the found string
            ' (which may have been a regular expression)
            ActiveDocument.Selection.CharRight

```

```

ActiveDocument.Selection.FindText str, _
    dsMatchBackward + dsMatchRegExp + dsMatchCase
nLen = Len(ActiveDocument.Selection)

ActiveDocument.Selection.CharLeft
if not ShouldIgnore then

    ' Now that we have the length, we need to redo
    ' the search and go on
    ActiveDocument.Selection.StartOfLine
    ActiveDocument.Selection.FindText "^.*" & str, _
        dsMatchRegExp + dsMatchCase

    bContinue = false

    ' If we're isolating a brace, make sure its matching brace
    ' isn't on the same line
    if (str = "{" or (str = "}") then
        ActiveDocument.Selection.CharRight
        nCurrentLine = ActiveDocument.Selection.CurrentLine
        nCurrentColumn = ActiveDocument.Selection.CurrentColumn
        ActiveDocument.Selection.CharLeft

        ExecuteCommand "GoToMatchBrace"
        if ActiveDocument.Selection.CurrentLine = nCurrentLine then
            ActiveDocument.Selection.MoveTo _
                nCurrentLine, nCurrentColumn
            bContinue = true ' we found it so move to the next match
        else
            ActiveDocument.Selection.MoveTo nCurrentLine, 1
            ActiveDocument.Selection.FindText "^.*" & str, _
                dsMatchRegExp + dsMatchCase
        end if
    end if

    if LTrimTabs(ActiveDocument.Selection) <> str and _
        not bContinue then
        ActiveDocument.Selection.CharRight
        MoveByColumns -nLen
        ActiveDocument.Selection.NewLine
        MoveByColumns nLen
    end if

    GetCurrentPosition strLastPos
end if
end if

wend

end sub

' Isolate the given str so that everything after it is placed on the next line
sub IsolateOnRight(str)
    dim strLastPos, strRight, nCurrentLine, nCurrentColumn, nLen

    strLastPos = InitializePosition

    while ActiveDocument.Selection.FindText(str & ".+$", _
        dsMatchRegExp + dsMatchCase) and GetCurrentPosition(strLastPos)

        ' Check if we're inside a comment or between quotes

```

```

ActiveDocument.Selection.CharLeft
if not ShouldIgnore then

    ' Repeat the search since ShouldIgnore puts the cursor at the
    ' beginning of the line
    ActiveDocument.Selection.FindText str & ".+$", _
        dsMatchRegExp + dsMatchCase

    ' Get the length of the found string
    ' (which may have been a regular expression)
    ActiveDocument.Selection.CharLeft
    ActiveDocument.Selection.FindText str, dsMatchRegExp + dsMatchCase
    nLen = Len(ActiveDocument.Selection)

    ' Now that we have the length, we need to redo the search and go on
    ActiveDocument.Selection.CharLeft
    ActiveDocument.Selection.FindText str & ".+$", _
        dsMatchRegExp + dsMatchCase

    strRight = LTrimTabs(Mid(ActiveDocument.Selection, nLen + 1))

    ' Handle braces a bit differently
    if (str = "{") or (str = ")") then

        ' If it's a closing brace, and the code after it contains
        ' a semicolon, leave it alone (ie. variable definition).
        if (str = ")") then
            ActiveDocument.Selection.EndOfLine dsExtend

        if (InStr(1, ActiveDocument.Selection, ";", vbTextCompare) > 0) then
            strRight = "" ' we found it so move on to the next match
        end if
        ActiveDocument.Selection.CharLeft
        ActiveDocument.Selection.CharRight
    end if

    ' If we're isolating a brace make sure the matching brace
    ' isn't on the same line
    if (strRight <> "") then
        ActiveDocument.Selection.CharLeft
        nCurrentLine = ActiveDocument.Selection.CurrentLine
        nCurrentColumn = ActiveDocument.Selection.CurrentColumn

        ExecuteCommand "GoToMatchBrace"

        if ActiveDocument.Selection.CurrentLine = nCurrentLine then
            ActiveDocument.Selection.MoveTo _
                nCurrentLine, nCurrentColumn + 1
            strRight = "" ' we found it so move on to the next match
        else
            ActiveDocument.Selection.MoveTo _
                nCurrentLine, nCurrentColumn
            ActiveDocument.Selection.FindText _
                str & ".+$", dsMatchRegExp + dsMatchCase
        end if
    end if
end if

if (strRight <> "") and _
    (Left(strRight, 1) <> "/" ) and _
    (strRight <> ",") and _

```

```

        (strRight <> ";") and _
        (strRight <> "\") then
            ActiveDocument.Selection.CharLeft
            MoveByColumns nLen
            ActiveDocument.Selection.NewLine
        end if

    end if

wend

end sub

' Place the given strOperator (of nLen REAL characters)
' between spaces (if needed)
sub FixOperator(strOperator, nLen)
    dim strLastPos, strFind

    strLastPos = InitializePosition

    ' Add one space between the operator
    while ActiveDocument.Selection.FindText("[A-Z,a-z,0-9,\),_,\]]" & _
        strOperator & "[A-Z,a-z,0-9,\(,_,\*,\"\",',&]", dsMatchRegExp) and _
        GetCurrentPosition(strLastPos)

        ' Check if we're inside a comment or between quotes
        ActiveDocument.Selection.CharLeft
        if not ShouldIgnore then

            ' Repeat the search since ShouldIgnore puts the cursor at the
            ' beginning of the line
            ActiveDocument.Selection.FindText "[A-Z,a-z,0-9,\),_,\]]" & _
                strOperator & "[A-Z,a-z,0-9,\(,_,\*,\"\",',&]", dsMatchRegExp

            ActiveDocument.Selection.CharLeft
            ActiveDocument.Selection.CharRight
            ActiveDocument.Selection = " "
            MoveByColumns nLen
            ActiveDocument.Selection = " "

        end if
    wend

    strLastPos = InitializePosition

    ' Fix any C++ "operator" member functions which were broken above
    while ActiveDocument.Selection.FindText("operator " & strOperator & " ", _
        dsMatchRegExp) and GetCurrentPosition(strLastPos)

        ' Check if we're inside a comment or between quotes
        if not ShouldIgnore then

            ' Repeat the search since ShouldIgnore puts the cursor at the
            ' beginning of the line
            ActiveDocument.Selection.FindText "operator " & strOperator & " ", _

        dsMatchRegExp
        ActiveDocument.Selection.CharRight
        ActiveDocument.Selection.Backspace
        MoveByColumns -nLen
        ActiveDocument.Selection.Backspace

    end if

```

```

wend

end sub

' Fix < operator without altering template<T> code and operator <<
function FixLessThanOperator()
    dim strLastPos, strFound, strTemplate

    strLastPos = InitializePosition

    while ActiveDocument.Selection.FindText("^.*[<]<.", dsMatchRegExp) and _
        GetCurrentPosition(strLastPos)

        ' Check if we're inside a comment or between quotes
        if not ShouldIgnore then

            ' Repeat the search since ShouldIgnore puts the cursor at the
            ' beginning of the line
            ActiveDocument.Selection.FindText "^.*[<]<.", dsMatchRegExp

            strFound = ActiveDocument.Selection

            ' Fix the left side
            strFound = Left(strFound, Len(strFound) - 2) & " " & _
                Right(strFound, 2)
            ActiveDocument.Selection = strFound

            ' Fix the right side
            strTemplate = Right(strFound, 11)
            if (Left(strTemplate, 8) <> "template") and _
                (Right(strFound, 1) <> " ") and _
                (Right(strFound, 1) <> "=") and _
                (Right(strFound, 1) <> "<") and _
                (Right(strFound, 1) <> ">") then
                ActiveDocument.Selection.CharLeft
                ActiveDocument.Selection = " "
            end if

        end if

    wend

end function

' Append a space after the given strOperator (if it needs it)
sub AppendSpace(strOperator)
    dim strLastPos

    strLastPos = InitializePosition

    while ActiveDocument.Selection.FindText(strOperator & _
        "[A-Z,a-z,0-9,\(\, \- ,_ ,\* ,\" ,' ,&]", dsMatchRegExp) and _
        GetCurrentPosition(strLastPos)

        ' Check if we're inside a comment or between quotes
        ActiveDocument.Selection.CharLeft
        if not ShouldIgnore then

            ActiveDocument.Selection.FindText strOperator & _
                "[A-Z,a-z,0-9,\(\, \- ,_ ,\* ,\" ,' ,&]", dsMatchRegExp

            ActiveDocument.Selection.CharLeft
            MoveByColumns Len(strOperator)

        end if

    wend

end sub

```



```

        ActiveDocument.Selection = " "

    end if
wend

end sub

' Fix tabbing within function blocks (surrounded by braces)
function TabifyMatchingBraces()
    dim strLastPos, cBeforeBrace

    strLastPos = InitializePosition

    while ActiveDocument.Selection.FindText("{") and _
        GetCurrentPosition(strLastPos)

        ' Check if we're inside a comment or between quotes
        if not ShouldIgnore then

            ' Repeat the action since ShouldIgnore puts the cursor at the
            ' beginning of the line
            ActiveDocument.Selection.FindText "{"

            ' Go to matching brace and reformat tabs
            ExecuteCommand "GoToMatchBraceExtend"
            ActiveDocument.Selection.SmartFormat

        if Len(ActiveDocument.Selection) > 1 then
            cBeforeBrace = Mid(ActiveDocument.Selection, _
                Len(ActiveDocument.Selection) - 1, 1)

            ' If SmartFormat indents the block (by mistake), unindent it
            if (cBeforeBrace = vbTab or cBeforeBrace = " ") then
                ActiveDocument.Selection.Unindent
            end if
        end if
    end if
wend

end function

' Since Microsoft's "SmartFormat" is not smart enough to indent case
' statements inside the switch body, we'll do it here.
' (Thanks to Jim Cooper)
function IndentSwitchBody()
    dim nSwitchLine, nFirstLine, nLastLine, strLastPos, iLine

    strLastPos = InitializePosition

    while ActiveDocument.Selection.FindText("switch", _
        dsMatchWord + dsMatchCase) and GetCurrentPosition(strLastPos)

        ' Check if we're inside a comment or between quotes
        if not ShouldIgnore then

            nSwitchLine = ActiveDocument.Selection.CurrentLine

            ' Now find the opening brace and make sure it's on the next line
            if ActiveDocument.Selection.FindText("{") and _
                not ShouldIgnore and _
                (ActiveDocument.Selection.CurrentLine = nSwitchLine + 1) then

```

```

        ' Repeat the action since ShouldIgnore puts the cursor at the
        ' beginning of the line
        ActiveDocument.Selection.FindText "{"

        ' Find next line in file, since earlier code put '{' on
        ' a line by itself
        nFirstLine = ActiveDocument.Selection.CurrentLine + 1

        ' Go to matching brace and reformat tabs
        ExecuteCommand "GoToMatchBrace"

        ' Find previous line in file, since earlier code put '}' on
        ' line by itself
        nLastLine = ActiveDocument.Selection.CurrentLine

        ' Move to the line after the opening brace
        ActiveDocument.Selection.GoToLine nFirstLine, 1

        ' Select everything between the braces and indent it
        for iLine = nFirstLine to nLastLine - 1
            ActiveDocument.Selection.LineDown dsExtend
        next
        ActiveDocument.Selection.Indent
    end if
end if
wend
end function

' Remove any lines that are considered extraneous (usually blank ones).
function RemoveExtraneousLines()
    dim strLastPos, nCurrentLine, nCurrentColumn

    strLastPos = InitializePosition

    ' Remove any blank lines that fall below any open braces ("{"")
    while ActiveDocument.Selection.FindText("{") and _
        GetCurrentPosition(strLastPos)

        ' Check if we're inside a comment or between quotes
        if not ShouldIgnore then
            ' Repeat the action since ShouldIgnore puts the cursor at the
            ' beginning of the line
            ActiveDocument.Selection.FindText "{"

            nCurrentLine = ActiveDocument.Selection.CurrentLine
            nCurrentColumn = ActiveDocument.Selection.CurrentColumn

            ActiveDocument.Selection.LineDown

            ' Cut any blank lines below the {
            do while true
                ActiveDocument.Selection.StartOfLine
                ActiveDocument.Selection.EndOfLine dsExtend

                if LTrimTabs(ActiveDocument.Selection) <> "" then
                    exit do
                end if
                ExecuteCommand "LineCut"

                ' Make sure we haven't hit the bottom of the file
                ActiveDocument.Selection.EndOfDocument
                if ActiveDocument.Selection.CurrentLine = nCurrentLine + 1 then

```

```

        exit do
    end if
    ActiveDocument.Selection.MoveTo nCurrentLine + 1, 1
loop
    ActiveDocument.Selection.MoveTo nCurrentLine, nCurrentColumn
end if
wend

strLastPos = InitializePosition

' Remove any blank lines right above any closing braces ("}")
while ActiveDocument.Selection.FindText("{}") and _
    GetCurrentPosition(strLastPos)

    ' Check if we're inside a comment or between quotes
    if not ShouldIgnore then
        ' Repeat the action since ShouldIgnore puts the cursor at the
        ' beginning of the line
        ActiveDocument.Selection.FindText "}"
        ActiveDocument.Selection.CharLeft

        ' Cut blank lines above the }
        do while true
            ActiveDocument.Selection.LineUp
            ActiveDocument.Selection.StartOfLine
            ActiveDocument.Selection.EndOfLine dsExtend

            if LTrimTabs(ActiveDocument.Selection) <> "" then

                if ActiveDocument.Selection.CurrentLine > 1 then
                    ActiveDocument.Selection.LineDown
                end if

                ActiveDocument.Selection.StartOfLine
                ActiveDocument.Selection.FindText "}"
                strLastPos = ActiveDocument.Selection.CurrentLine & "," & _
                ActiveDocument.Selection.CurrentColumn
                ActiveDocument.Selection.LineDown
                ActiveDocument.Selection.StartOfLine
                exit do
            end if
            ExecuteCommand "LineCut"
        loop
    end if
wend

end function

' Remove all spaces and tabs found in the current selection
function RemoveSpacesInSelection
    dim iPos, ch, strNoSpaces

    for iPos = 1 to Len(ActiveDocument.Selection)
        ch = Mid(ActiveDocument.Selection, iPos, 1)
        if ch <> " " and ch <> vbTab then
            strNoSpaces = strNoSpaces + ch
        end if
    next

    ActiveDocument.Selection = strNoSpaces
end function

```

```

' Fix any code with exponential notation (ie. 3.4e-2) which was
' broken when the + and - operators were fixed above (by FixOperator).
function FixExponents

    while ActiveDocument.Selection.FindText("[0-9,\.]e [\+!\-] [0-9]", _
        dsMatchRegExp)
        RemoveSpacesInSelection
    wend

end function

' Break any lines containing multiple statements (separated by semicolons)
function BreakLinesWithMultipleStatements()
    dim strLastPos, nCurrentLine

    strLastPos = InitializePosition

    ' Search for multiple semicolons on the same line
    while ActiveDocument.Selection.FindText(";+;", dsMatchRegExp) and _
        GetCurrentPosition(strLastPos)

        ' Check if we're inside a comment or between quotes
        if not ShouldIgnore then
            ' Repeat the action since ShouldIgnore puts the cursor at the
            ' beginning of the line
            ActiveDocument.Selection.FindText ";+;", dsMatchRegExp

            nCurrentLine = ActiveDocument.Selection.CurrentLine
            ActiveDocument.Selection.CharLeft
            ActiveDocument.Selection.StartOfLine dsFirstText, dsExtend

            ' If found, check that the semicolons don't belong to a for loop
            if (InStr(1, ActiveDocument.Selection, "for (", _
                vbTextCompare) > 0) then
                ActiveDocument.Selection.MoveTo nCurrentLine + 1, 1
            else
                ActiveDocument.Selection.CharRight
                ActiveDocument.Selection.CharRight
                ActiveDocument.Selection.NewLine
            end if
        end if
    wend

end function

' Make sure the first C++ comment of every line has a space after it.
function InsertSpaceAfterFirstInLineComment()
    dim strLastPos, nCurrentLine

    strLastPos = InitializePosition

    while ActiveDocument.Selection.FindText("//[A-Z,a-z,0-9]", dsMatchRegExp) _
        and GetCurrentPosition(strLastPos)

        ActiveDocument.Selection.CharLeft

        ' Check if we're inside a comment or between quotes
        if not ShouldIgnore then
            ' Repeat the action since ShouldIgnore puts the cursor at the
            ' beginning of the line

```

```

        ActiveDocument.Selection.FindText "[A-Z,a-z,0-9]", dsMatchRegExp

        ActiveDocument.Selection.CharRight
        ActiveDocument.Selection.CharLeft
        ActiveDocument.Selection = " "
    end if
wend

end function

Sub MakeSelectedCodeNicer()
'DESCRIPTION: Reformats the currently selected source code to look nicer.
' Written by Alvaro Mendez on 07/15/1999
' Last Updated on 11/01/1999

    ' Check if there's a valid selection
    if ActiveDocument.Selection = "" then
        exit sub
    end if

    ' Copy the selection to the clipboard
    ActiveDocument.Selection.Copy

    ' Open a new document and changed its language to C/C++
    ' This is required for SmartIndent to work.
    Documents.Add "Text"
    ActiveDocument.Language = "C/C++"

    ' Paste the selection into the document and run the macro
    ActiveDocument.Selection.Paste
    ExecuteCommand "MakeCodeNicer"

    ' Select the resulting code and copy it to the clipboard
    ActiveDocument.Selection.SelectAll
    ActiveDocument.Selection.Copy

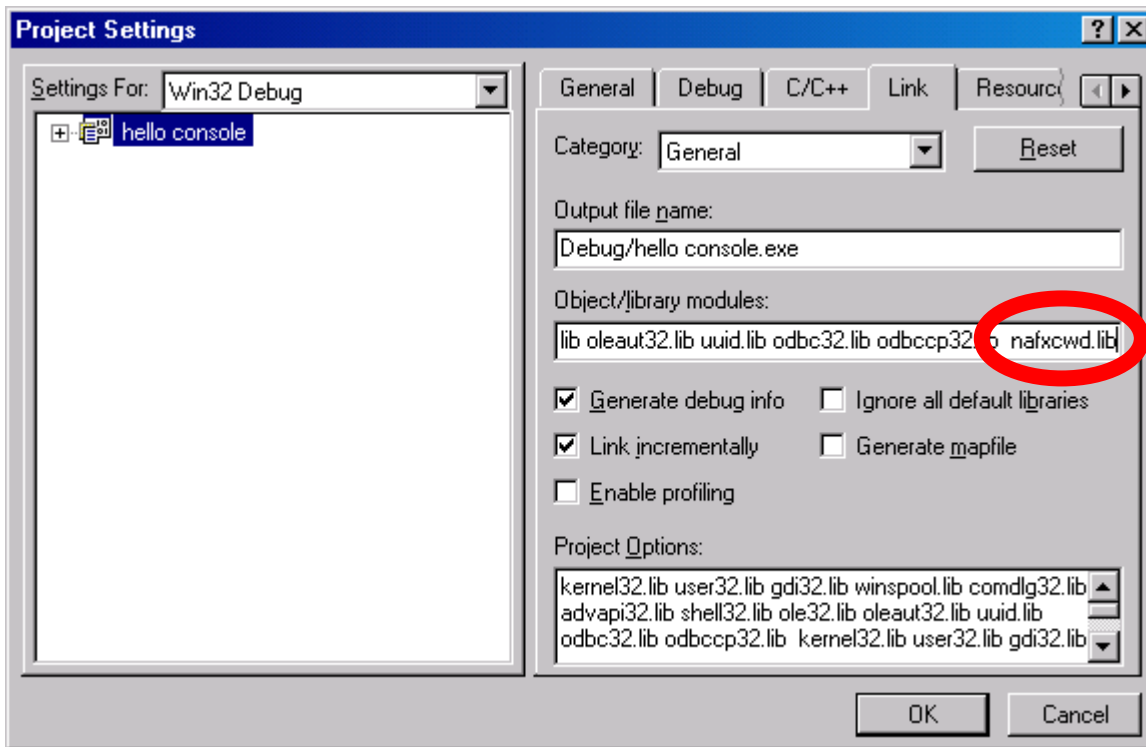
    ' Close the new document (without saving it)
    ActiveWindow.Close dsSaveChangesNo

    ' Paste the reformatted code back over the original selection
    ActiveDocument.Selection.Paste
End Sub

```

4.1.7.2 Acrescentando Lib no Project Settings

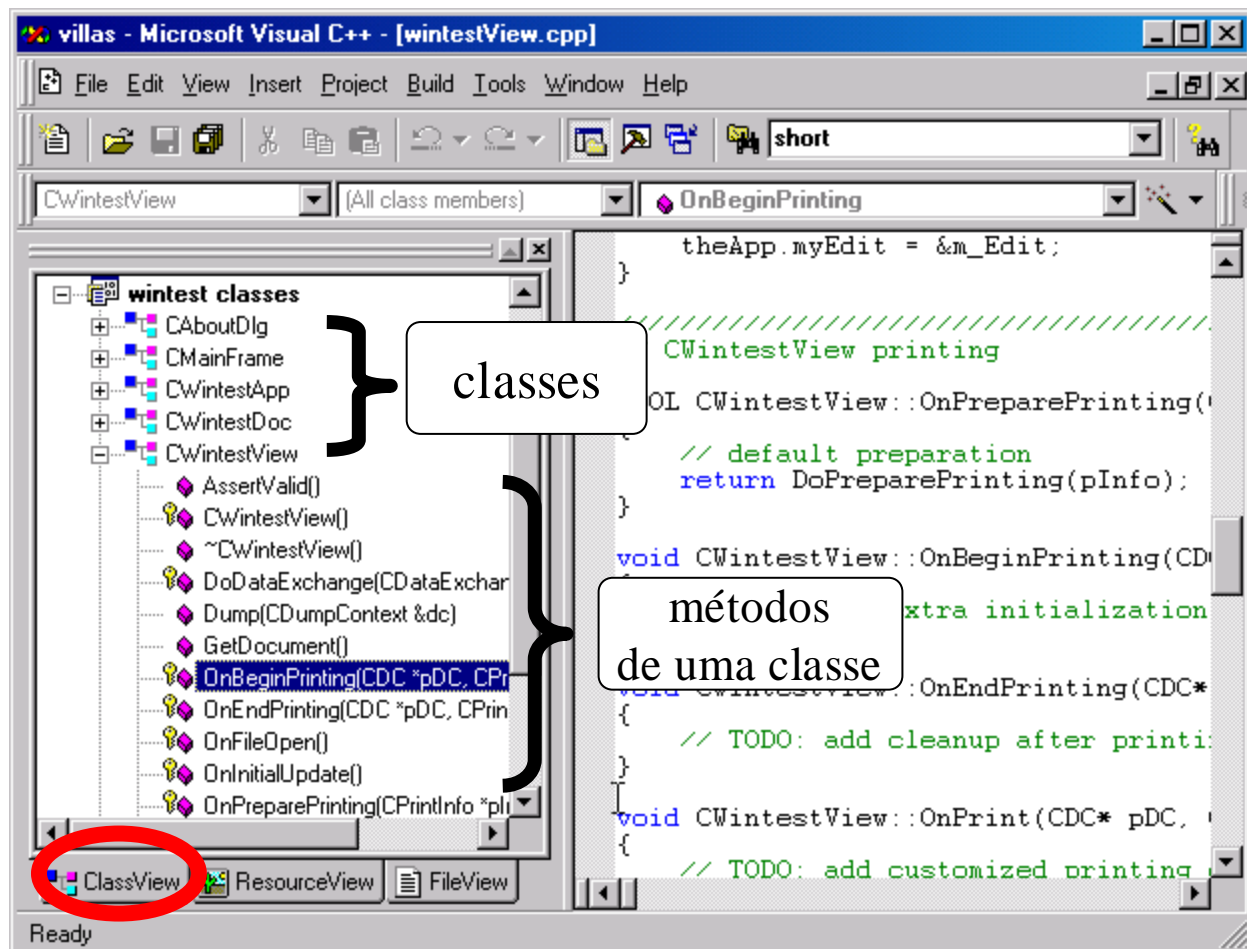
Ao usar certas funções, é preciso incluir a biblioteca correspondente. Por exemplo, para se usar a função “GetCurrentDirectory”, é preciso se incluir a biblioteca “nafxcwd.lib”. Para acrescentar uma biblioteca a ser linkada no projeto, comande Projects - Settings (<alt>F7)- <Link tab> - no campo de <Object/library modules>, clique no campo e acrescente no final da linha o nome da biblioteca a acrescentar “nafxcwd.lib”.



4.1.7.3 Class View

A funcionalidade “Class View” do Visual C++ é um interessante diferencial desse produto. A idéia é “enxergar” um projeto de um programa C++ a partir da descrição de suas classes, mais que pela descrição de seus arquivos (o que corresponde a “File View”).

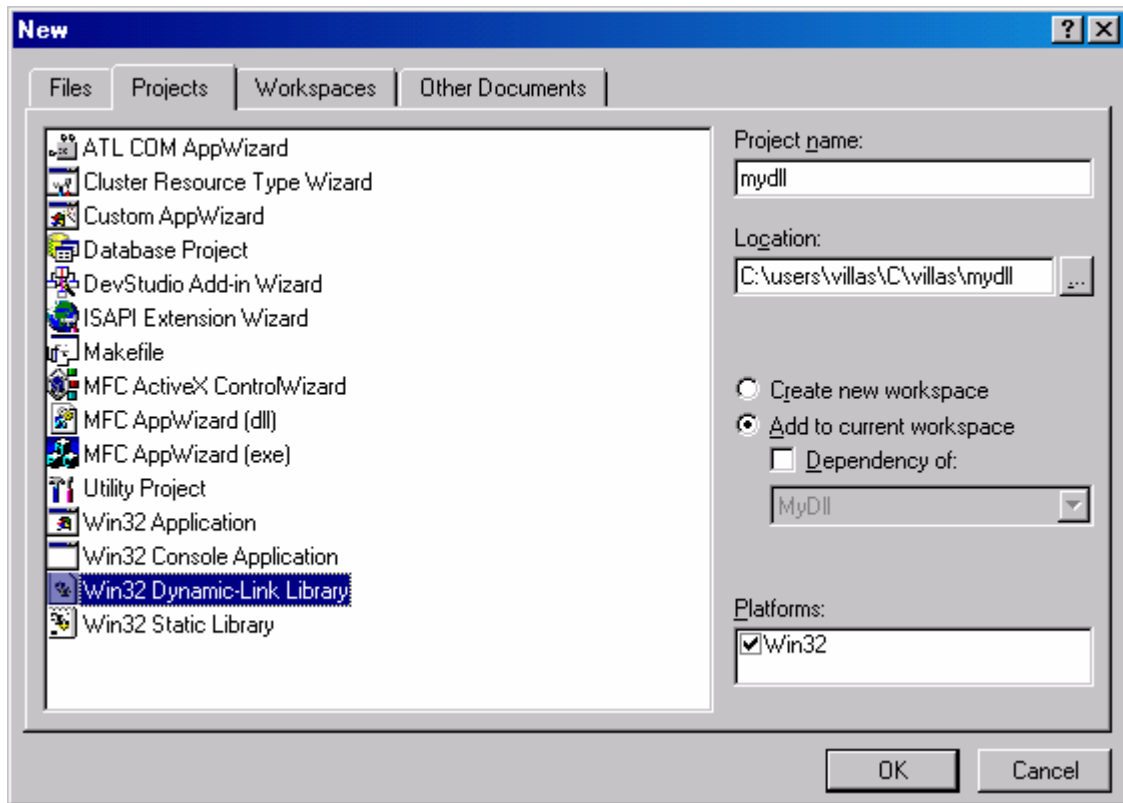
Há inúmeras funcionalidades de navegação no código C++ a partir da funcionalidade do Class View. Por exemplo: clicando-se num método de uma classe, automaticamente o editor é aberto e o cursor posicionado na classe em questão.



4.1.7.4 Usando bibliotecas de ligação dinâmica (DLL)

Uma biblioteca de ligação dinâmica ("Dynamic Link Library - DLL") é uma biblioteca que será ligada ao programa executável em tempo de execução. Para fazer uma dll, a primeira coisa a fazer é um projeto novo, em que se escolhe "Win32 Dynamic-Link Library", como mostrado na figura. A dll deve ter um nome (no caso "mydll", e pertencer a algum workspace). No wizard, escolha "An empty DLL project".

Abaixo pode-se ver um exemplo de dll. Nesse exemplo, a biblioteca contém 3 funções globais. Uma é `dllTest`, outra é `plus_1`, e a terceira é `plus_2`. A `dllTest` apenas coloca uma constante string no console. As outras duas, recebem um inteiro como parâmetro e retornam um inteiro (somando 1 e 2 respectivamente). São funções simples de teste, úteis para um tutorial.



No arquivo mydll.cpp, escreve-se o corpo da dll, acrescentando-se a macro WINAPI ao protótipo da função. As regras de compilação da dll são as mesma de qualquer arquivo cpp.

```

////////////////////////////////////
// mydll.cpp
#include <iostream.h>
#include "mydll.h"
void WINAPI dllTest() {
    cout << "You're inside a Dll" << endl;
}
int WINAPI plus_1(int i) {
    return i+1;
}
int WINAPI plus_2(int i) {
    return i+2;
}

```

No arquivo mydll.h, coloca-se apenas os protótipos das funções globais da dll, acrescentando-se a mesma macro WINAPI. Uma outra macro _MYDLL_H_ controla a possibilidade de inclusão múltipla do header mydll.h.

```

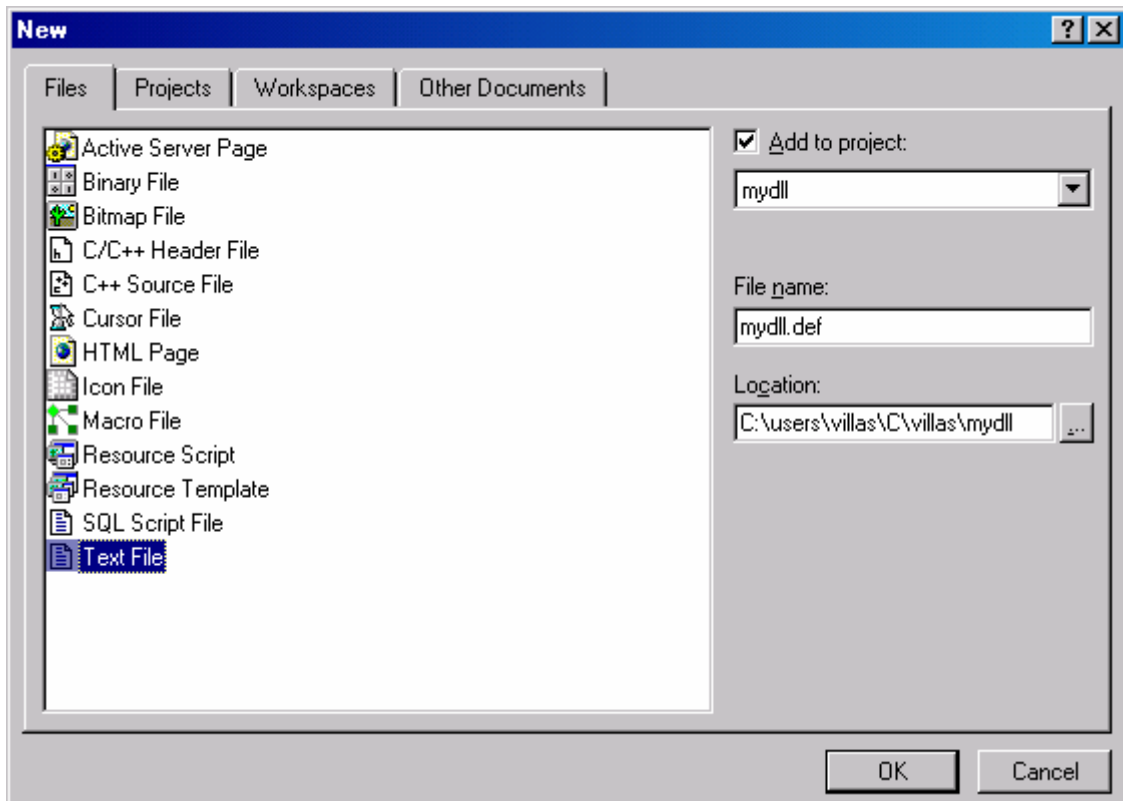
////////////////////////////////////
// mydll.h
#ifndef _MYDLL_H_
#define _MYDLL_H_
#include <afxwin.h>
// place the prototypes of dll functions
void WINAPI dllTest();
int WINAPI plus_1(int i);
int WINAPI plus_2(int i);

```

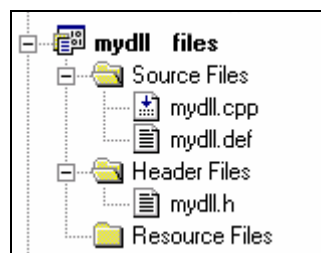


```
#endif // _MYDLL_H_
```

Além do cpp e do header, é preciso que exista um arquivo texto de definição. No caso esse arquivo é mydll.def. O modelo abaixo mostra como fazer esse arquivo. Além do nome e descrição, é preciso que se faça uma lista dos identificadores a serem exportados (no caso o nome das 3 funções globais). Para criar o arquivo def, basta escolher no menu “File-New” e no tab “files” escolher o nome com terminação *.def.



O arquivo mydll.def aparece no projeto, como se fosse um arquivo *.cpp.



```

////////////////////////////////////
; mydll.def : Declares the module parameters for the DLL.
LIBRARY      "mydll"
DESCRIPTION  'MyDll Windows Dynamic Link Library'
EXPORTS
; Explicit exports can go here
dllTest @1
plus_1 @2
plus_2 @3

```

Para usar a dll, é preciso fazer a seguinte sequência de operações. Em primeiro lugar é preciso carregar a dll. Isso é feito com a função `LoadLibrary`. Essa operação pode falhar (como a abertura de um arquivo). Portanto, um programa bem feito deverá tratar a possibilidade de o carregamento da dll falhar.

O programa deve ter variáveis tipo “ponteiro para função”, para receber as funções da dll. Para usar as funções, a segunda coisa a fazer é ligar os ponteiros para função locais com as funções na dll. Isso é feito com a função `GetProcAddress`. Novamente, essa operação pode falhar e um programa bem feito deve tratar a possibilidade de falha. Repare que o retorno da função `GetProcAddress` deve ser maquiado (*type cast*) para ser compatível com a variável local tipo “ponteiro para função”.

Com os ponteiros para função carregados, basta usa-los como se fossem funções do próprio programa. Note que como o carregamento da dll é feito em tempo de execução, seria bastante simples que alguma lógica escolhesse carregar uma ou outra dll equivalente. Por exemplo: a dll poderia conter mensagens ou recursos, em que as frases seriam escritas em um determinado idioma. Assim um software poderia ser internacionalizado apenas por escrever uma nova dll (e fazer o programa principal encontrar essa dll, o que seria feito na instalação do programa).

Para conseguir ligar (linkar) o `usedll.cpp`, é preciso acrescentar a biblioteca “`nafxcwd.lib`”. Veja a seção 4.1.7.1, na página 108. No caso desse exemplo, foi feito um novo projeto, tipo “win32 console application”, chamado `usedll`. Portanto, considerando que as saídas dos projetos ficam no diretório “Debug”, ou “Release”, a forma de apontar a dll a partir do projeto `usedll` é a string “`..\..\mydll\Debug\mydll.dll`”. Uma dll pode ser executada sob debug tanto quanto um programa normal, desde que tenha sido compilada para debug. Também como um programa ou biblioteca normal, caso não haja interesse de debug (e.g. entregando o arquivo para um cliente), é recomendável compilar como Release.

```
////////////////////////////////////
// usedll.cpp (mydll.dll)
// use_dll.cpp
#include <iostream.h>
#include <afxwin.h>
int main() {
    HINSTANCE hDll; // dll handler
    hDll = LoadLibrary("../..\mydll\Debug\mydll.dll"); // load dll

    // handle error while loading dll handler
    if (!hDll) {
        cout << "Error while loading DLL handler" << endl;
        exit(1);
    }

    // function pointers equivalent to dll functions
    void (WINAPI * c_dllTest)();
    int (WINAPI *c_plus_1)(int);
```

```

int (WINAPI *c_plus_2)(int);

// link function pointers to dll functions
c_dllTest = (void (WINAPI *)())GetProcAddress(hDll, "dllTest");
c_plus_1 = (int (WINAPI *)(int))GetProcAddress(hDll, "plus_1");
c_plus_2 = (int (WINAPI *)(int))GetProcAddress(hDll, "plus_2");

// handle error while loading functions
if(!c_dllTest || !c_plus_1 || !c_plus_2) {
    cout << "Error while loading functions from DLL" << endl;
    exit(1);
}

// now, use the functions freely
c_dllTest();
cout << "3+1=" << c_plus_1(3) << endl;
cout << "5+2=" << c_plus_2(5) << endl;
return 0;
}

```

Saída do programa:

```

You're inside a Dll
3+1=4
5+2=7

```

4.1.7.5 DLL para Windows

Tipos de dll com Visual C para Windows - Projeto “MFC AppWizard (dll)”.

1. Regular DLL with MFC static linked
2. Regular DLL using shared MFC DLL
3. MFC Extension DLL (using shared MFC DLL)

A dll tipo “regular” (caso 1. e 2.) pode em princípio ser usada em outra linguagem que não Visual C++ (como Delphi). A dll tipo “extension” somente pode ser usada para programas com Visual C++.

4.1.7.6 Otimização do linker para alinhamento de código

Na versão 6.0 do Visual C++, há uma opção de otimização do linker que é “/OPT:WIN98”, que é definida como verdadeira implicitamente. Com essa definição, o alinhamento do código é feito com 4K de tamanho (sem essa opção ativa, o alinhamento é de 512 bytes). O alinhamento em 4K torna o código um pouco maior. Pode-se comandar esse flag diretamente pelo código usando #pragma. No exemplo abaixo, o flag é resetado (desativado). Esse código deve ser acrescentado no início do código fonte em questão.

```
#pragma comment(linker, “/OPT:NOWIN98”)
```

4.1.8 Detectando vazamento de memória

Para detectar vazamento de memória, é preciso duas funcionalidades:

1. Salvar a condição da memória alocada

2. Checar a condição corrente da memória alocada. Se for igual a que está salva, conclui-se que entre o momento que se salvou e o momento que se checkou não houve vazamento de memória. Ao contrário, se não for igual, conclui-se que houve vazamento de memória.

Uma forma prática de lidar com o problema de vazamento de memória é escrever uma classe para essa finalidade. Abaixo está escrita essa classe específica para o compilador Visual C++ 6.0. A classe `vb_memCheck` possui apenas 2 métodos. O construtor salva a condição de memória alocada. O outro método é `check`, que checka o vazamento e manda para o console informação de vazamento de memória caso haja algum. Se não houver vazamento, o método `check` não faz nada.

```
#include <crtdbg.h> // include necessary to handle memory leak debug (win32 only)

// use this class for Visual C++ 6.0 only !!
class VbMemCheck {
    _CrtMemState s1;
public:
    VbMemCheck() { // store memory on constructor
        // Send all reports to STDOUT
        _CrtSetReportMode( _CRT_WARN, _CRTDBG_MODE_FILE );
        _CrtSetReportFile( _CRT_WARN, _CRTDBG_FILE_STDOUT );
        _CrtSetReportMode( _CRT_ERROR, _CRTDBG_MODE_FILE );
        _CrtSetReportFile( _CRT_ERROR, _CRTDBG_FILE_STDOUT );
        _CrtSetReportMode( _CRT_ASSERT, _CRTDBG_MODE_FILE );
        _CrtSetReportFile( _CRT_ASSERT, _CRTDBG_FILE_STDOUT );

        // Store memory checkpoint in s1 (global variable) memory-state structure
        _CrtMemCheckpoint( &s1 );
    };

    void check() {
        _CrtDumpMemoryLeaks();
    };
};
```

No exemplo de utilização abaixo, não há vazamento de memória.

```
int main () {
    cout << "hello" << endl;

    VbMemCheck c; // construtor salva condição da memória

    int *p_i=new int; // código que pode ter vazamento de memória
    delete p_i;

    c.check(); // manda aviso para console caso haja vazamento
    return 0;
}
```

Nesse caso, o resultado é:

hello

Caso haja vazamento de memória, como no caso abaixo,

```
int main () {
    cout << "hello" << endl;

    VbMemCheck c; // construtor salva condição da memória
```

```

    int *p_i=new int; // código que pode ter vazamento de memória

    c.check(); // manda aviso para console caso haja vazamento

    return 0;
}

```

O resultado é:

```

hello
Detected memory leaks!
Dumping objects ->
{20} normal block at 0x00780DA0, 4 bytes long.
    Data: <      > CD CD CD CD
Object dump complete.

```

4.2 Borland C++ builder 5.0

Essa versão do compilador C++ da Borland foi lançada no ano de 2000. Trata-se de um produto com muitas funcionalidades. A mesma Borland possui também um produto chamado Delphi, que é uma variante de Pascal, e é considerado um RAD (*Rapid Application Environment*, ou ambiente de [desenvolvimento] rápido de aplicação) muito aceito no mercado. O Borland builder é uma tentativa de refazer o ambiente RAD do Delphi em C++. As semelhanças entre o ambiente gráfico do Delphi e do builder são óbvias.

Com respeito a programação para Windows, a Borland já recomendou o uso da biblioteca OWL. Posteriormente, já há algum tempo, vem recomendando o uso de outra biblioteca chamada VCL. Tanto OWL quanto VCL são de autoria da Borland. É interessante lembrar que a Microsoft recomenda o uso da biblioteca MFC, de sua própria autoria. Não é viável na prática desenvolver programas para Windows em C++ sem uma biblioteca de apoio. A decisão de qual biblioteca usar é estratégica, pois trata-se de algo que requer tempo para se aprender. Por várias razões, principalmente por razões estratégicas e de mercado, eu recomendo o uso de MFC (que pode ser usado tanto pelo compilador da Borland quanto pelo compilador da Microsoft).

Nessa seção, fala-se de como usar o Borland C++ builder para programas DOS.

4.2.1 Reconhecendo o Compilador

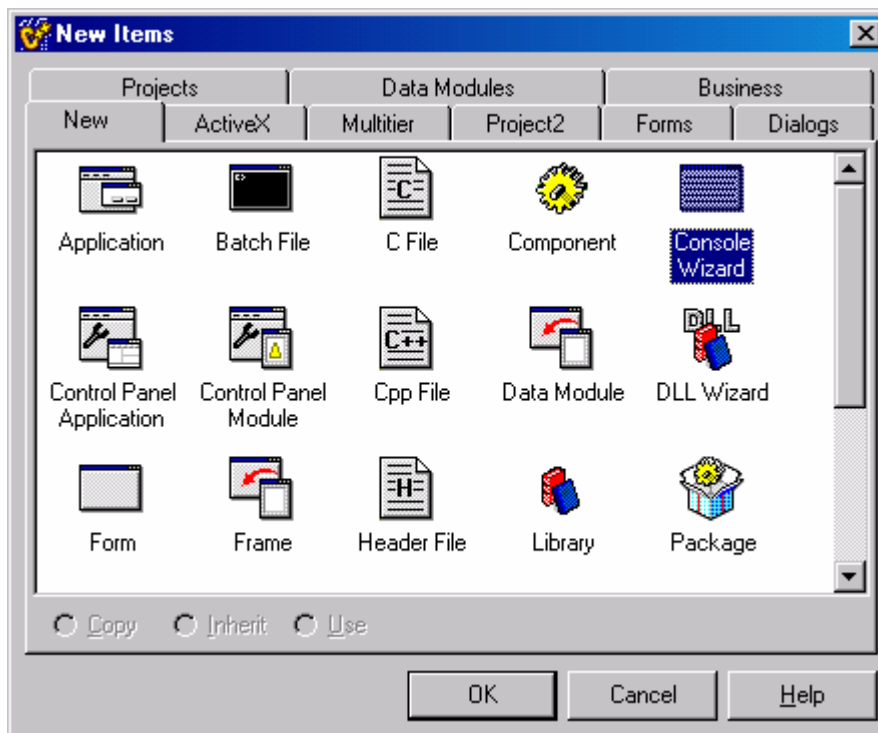
O builder 5 é um programa que não usa a estrutura (*framework*) padrão de SDI nem de MDI. O programa em si é uma pequena janela apenas (na forma padrão) com barra de título (*title bar*), menu, e duas barras de ferramentas (*toolbars*) (veja figura abaixo). Pode-se editar mais de um arquivo ao mesmo tempo, por isso o programa é de certa forma parecido com a estrutura MDI. Geralmente, ao abrir-se o builder 5, automaticamente outras janelas de edição são automaticamente inicializadas. Mas nesse tutorial se está partindo do programa

em si (considerando que os arquivos que foram abertos automaticamente foram fechados).

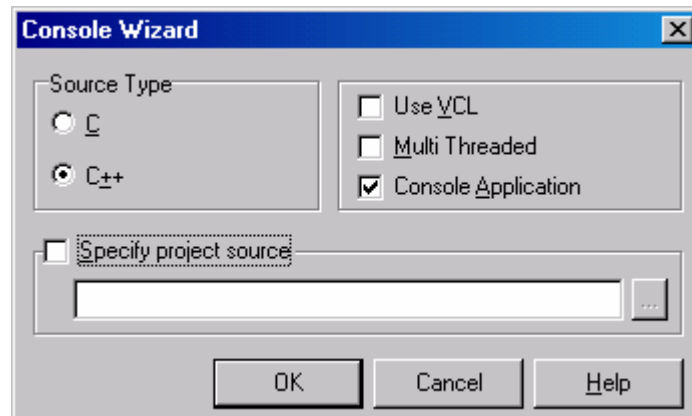


4.2.2 "Hello world" para DOS

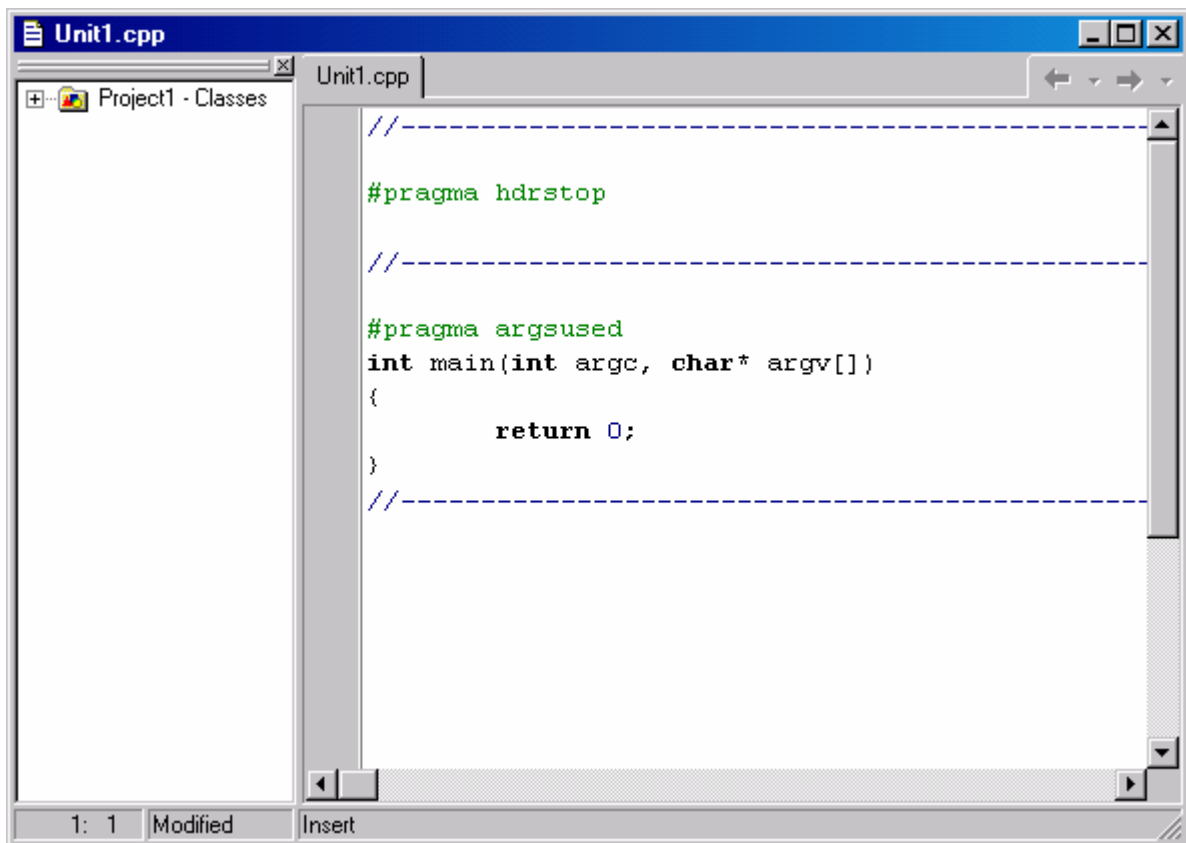
Para fazer um programa para console com builder 5, escolha no menu File-New. Na caixa de diálogo "New Items", escolha "console Wizard".



Na caixa de diálogo "Console Wizard", escolha as opções como mostrado na figura abaixo.



O builder então cria um programa de esqueleto, num arquivo chamado Unit1.cpp, como mostrado abaixo. Já existe um programa mínimo nesse arquivo fonte.



Apesar da recomendação da Borland, eu recomendo que se apague o conteúdo do arquivo e substitua-o pelo texto abaixo.

```
#include <iostream.h>  
int main() {  
    cout << "Hello, world" << endl;  
    return 0;  
}
```

A resposta do programa é como mostrado abaixo.

```
Hello, world
```

Mas como o programa acaba, haverá se poderá perceber uma janela piscando rapidamente na tela. Para que a janela fique esperando uma tecla ser apertada, mude o programa para o texto abaixo. Nessa versão, foi acrescentada a linha “cout << “Press any key”; getch();” no final, e também “#include <conio.h>” no início. Assim, o programa ficará sempre esperando uma tecla para terminar (como o Visual C++).

```
#include <conio.h>
#include <iostream.h>
int main() {
    cout << "Hello, world" << endl;
    cout << "Press any key"; getch();
    return 0;
}
```

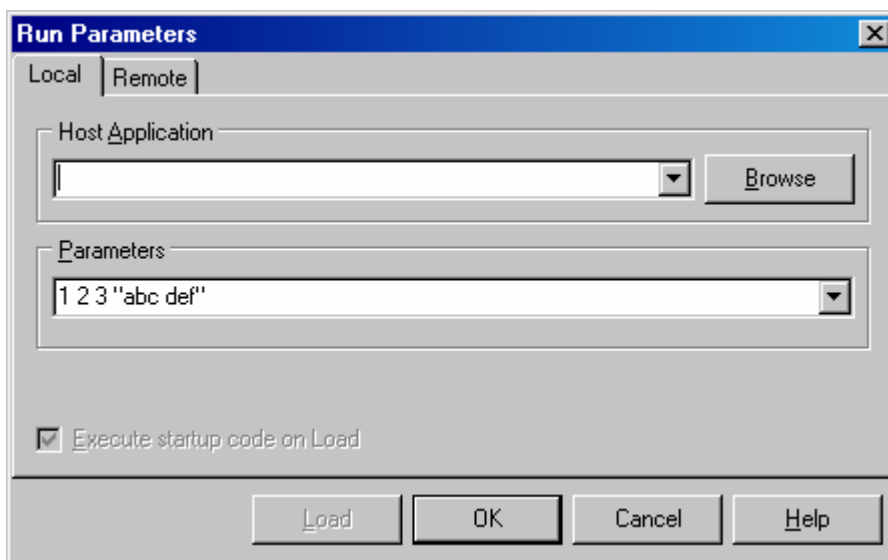
A resposta do programa é como mostrado abaixo.

```
Hello, world
Press any key
```

Nos exemplos que se seguem para builder 5, a linha “cout << “Press any key”; getch();” não será mais mencionada, ficando o seu uso a critério do leitor.

4.2.2.1 Adicionando argumentos para a linha de comando

No menu, selecione Run-Parameters. Surge a caixa de diálogo “Run Parameters” como mostrado abaixo.



No caso dos parâmetros mostrados acima, rodando-se o programa abaixo,

```
#include <iostream.h>
int main(int argc, char**argv) {
    cout << "Hello, world\n";
    int i;
    for (i=0; i<argc; i++)
        cout << "Arg[" << i << "]: " << argv[i] << endl;
    return 0;
}
```


obtém-se o resultado como mostrado abaixo.

```
Hello, world
Arg[0]:C:\Program Files\Borland\CBuilder5\Projects\Project1.exe
Arg[1]:1
Arg[2]:2
Arg[3]:3
Arg[4]:abc def
```

4.3 C++ para win32 gratuito

4.3.1 MinGW

O compilador MinGW é gratuito, baseado no compilador da GNU e funciona em DOS (também em Windows). A página web desse compilador é [51].

4.3.2 djgpp

A página do djgpp é www.delorie.com/djgpp/. Nessa página, há todas as informações necessárias para copiar, instalar e usar esse compilador. Abaixo há um resumo das informações de instalação.

O uso do djgpp é muito parecido com o do g++ do unix.

Pegue num dos servidores ftp listados, no diretório v2gnu, os arquivos abaixo (pequenas variações nas versões dos arquivos em princípio são aceitas). As descrições do significado desses arquivos estão disponíveis na página do djgpp.

- bnu2951b.zip
- djdev203.zip
- gcc2953b.zip
- gpp2953b.zip
- txi40b.zip

Descompacte todos esses arquivos para um diretório de instalação. Digamos que esse diretório de trabalho. Digamos que o nome desse diretório seja c:\winap\djgpp. Para simplificar, se quiser, crie um arquivo chamado djgpp.bat com o conteúdo abaixo.

```
@echo off
set PATH=C:\winap\DJGPP\BIN;%PATH%
set DJGPP=c:/winap/djgpp/djgpp.env
echo DJGPP enviromnent installed (sbVB)
```

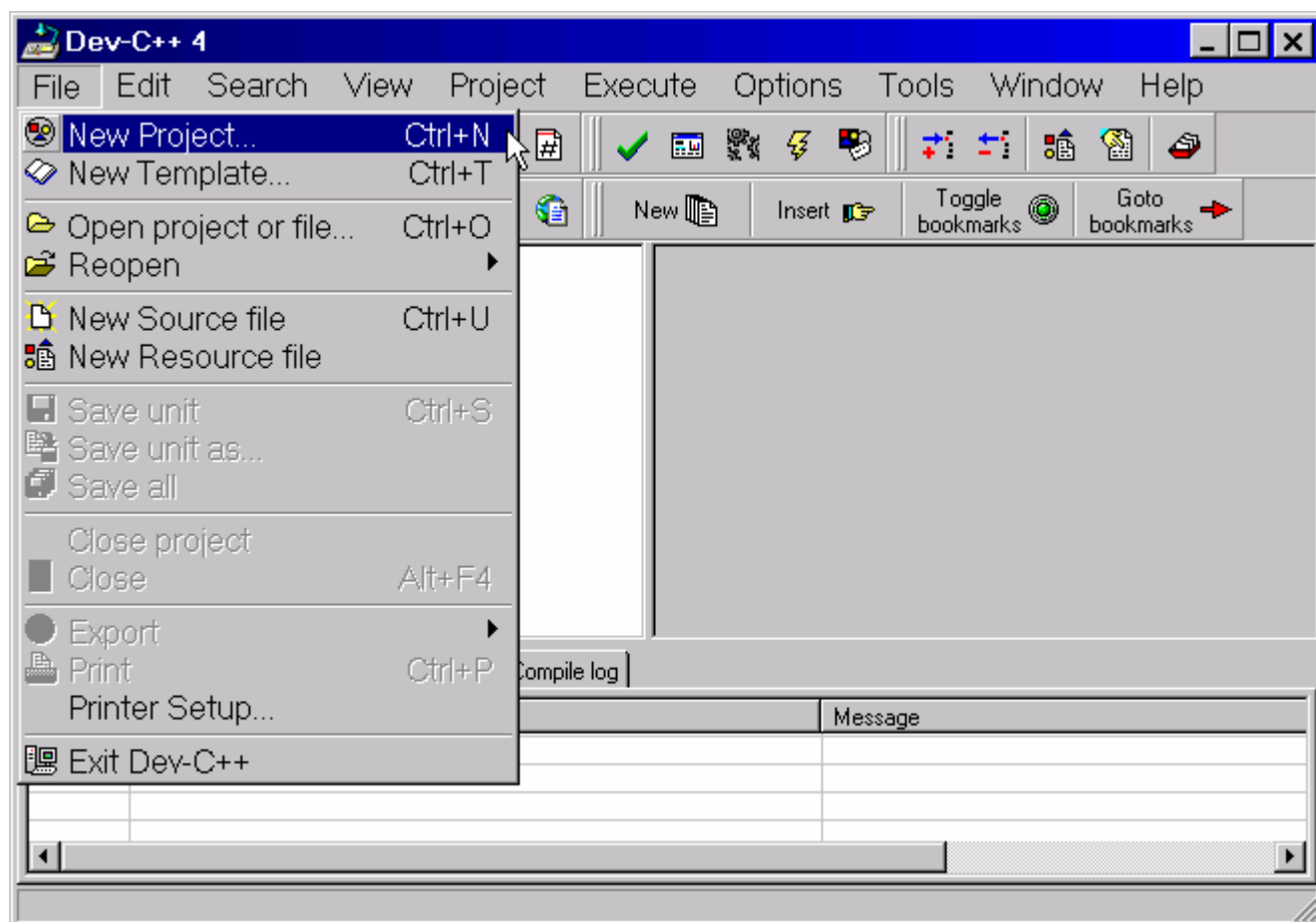
Um problema dessa versão de compilador é que muitas bibliotecas são acrescentadas no programa executável final. Com isso mesmo um programa pequeno (tipo “hello world” fera um executável grande (maior que 300K).

4.4 Dev-C++ 4.0

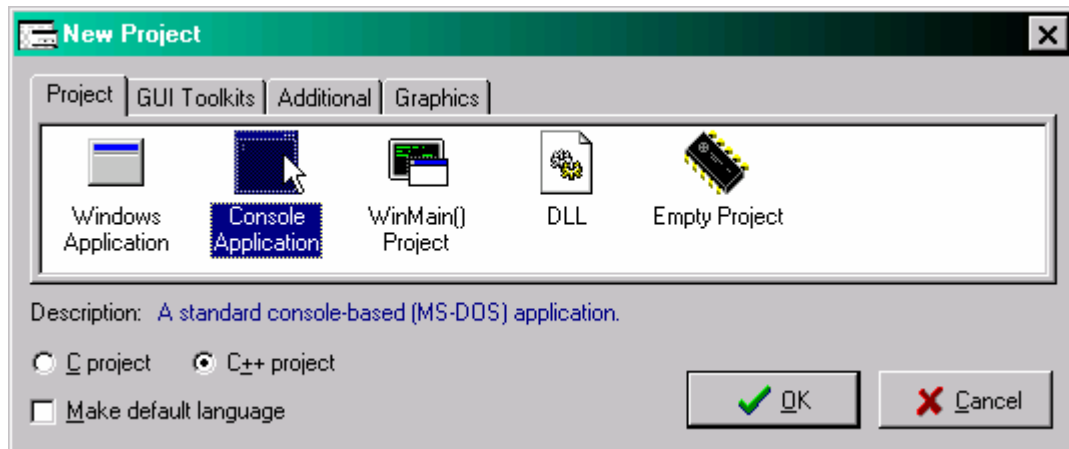
O Dev-C++ é um software gratuito. Na página do distribuidor [47], ou em um site de download, procurando-se por “dev-c” (por exemplo [48]) pode-se obter uma cópia desse compilador. Trata-se de um compilador baseado no MinGW [51], que é uma versão win32 do g++ do unix. No Dev-C++ 4.0 fez-se um aplicativo GUI com boa sofisticação, e com isso facilita-se muito a usabilidade.

4.4.1 Reconhecendo o Compilador

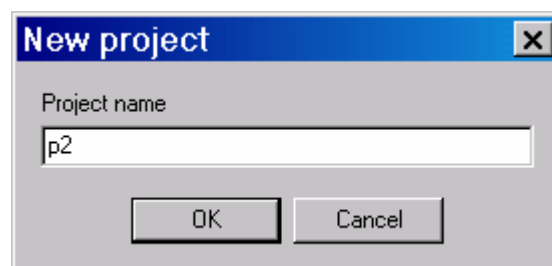
O Dev-C++ trabalha com projetos (*.dev). Cada projeto é a descrição de como um conjunto de arquivos fonte faz gerar um executável. Para gerar um executável para console, selecione no menu principal: File - New Project ... (<ctrl-N>), como mostrado na figura abaixo.



Selecione “Console Application”, e “C++ project”, como mostrado na caixa de diálogo abaixo.

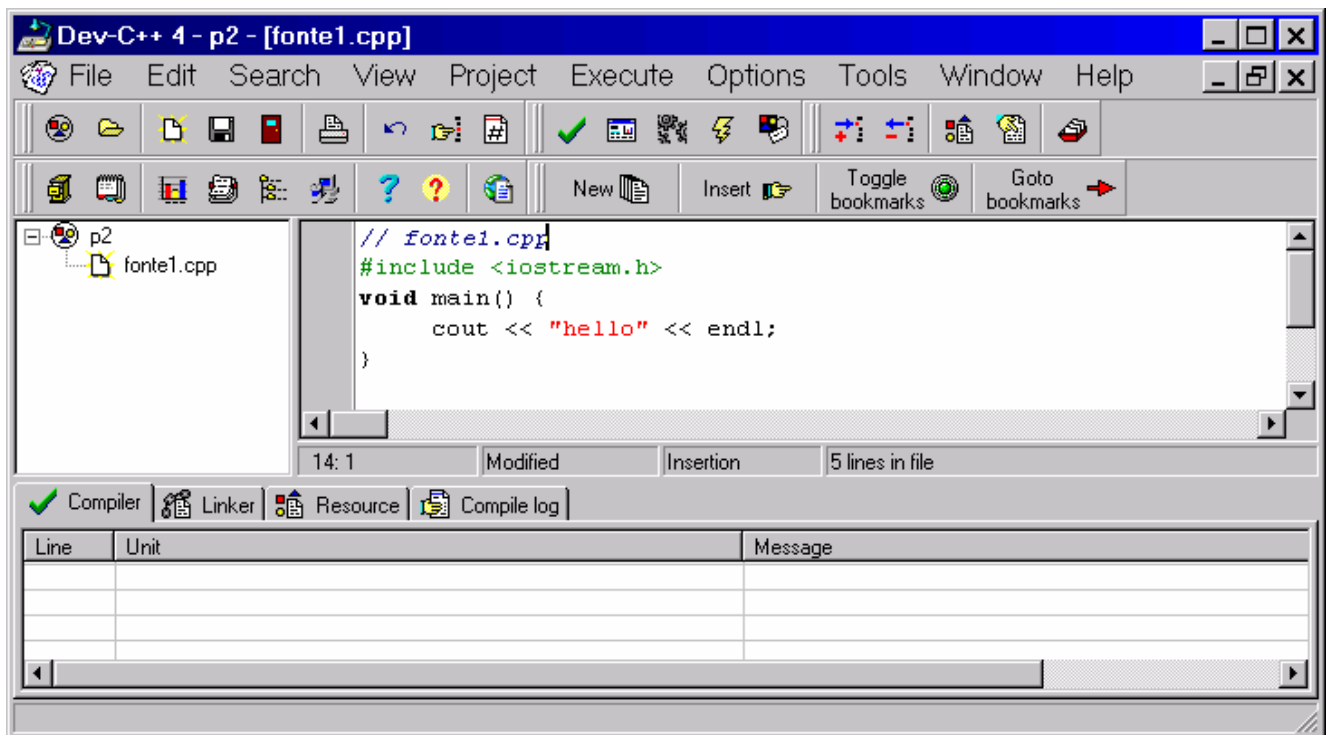


Dê um nome para o projeto. No caso, o nome dado foi “p2”. Com isso, será gerado o arquivo de projeto “p2.dev”. O executável após a construção (compilação e ligação) se chamará “p2.exe”.



4.4.2 “Hello world”

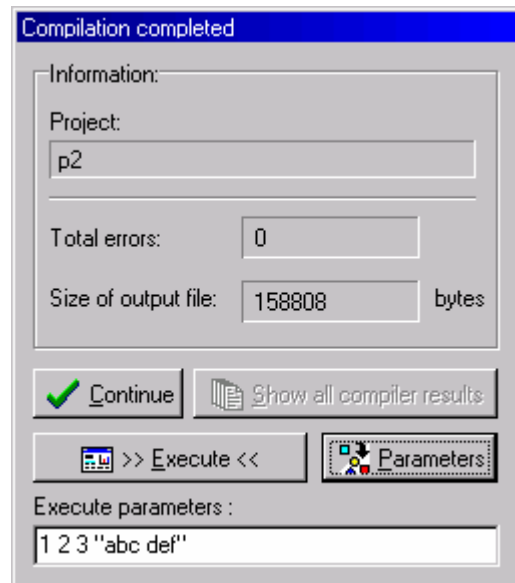
Crie o arquivo de texto fonte1.cpp (veja na página 99). O próprio ambiente integrado edita o arquivo. Veja a figura abaixo.



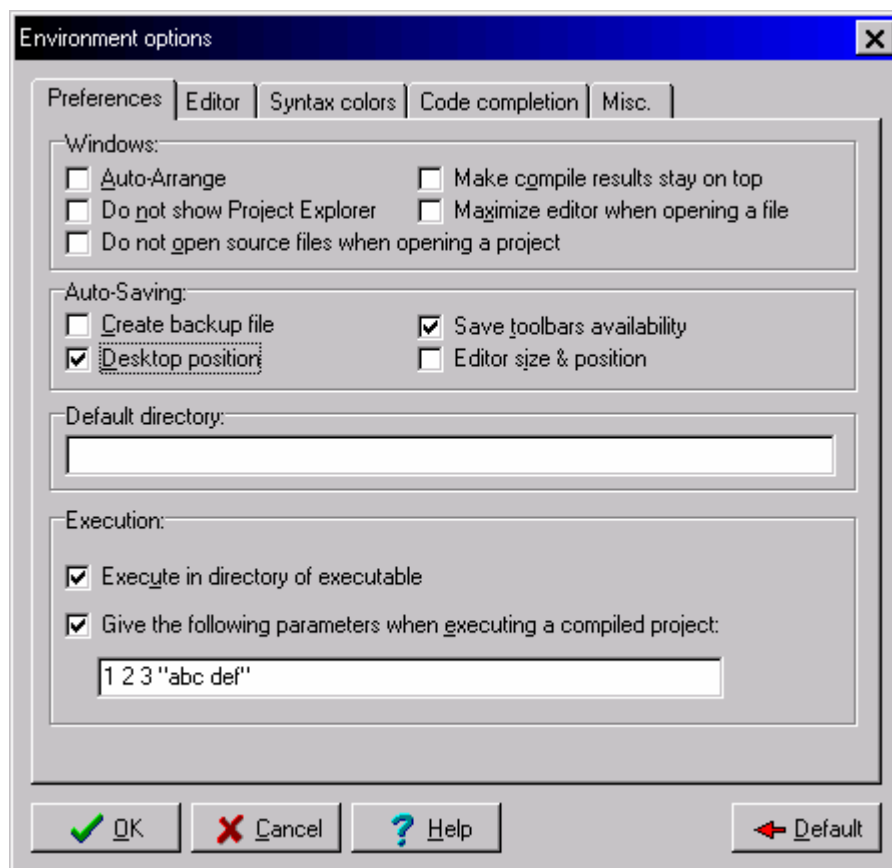
4.4.2.1 Adicionando argumentos para a linha de comando

Chamar um programa passando argumentos pela linha de comando faz particularmente muito sentido no caso de compiladores que tem interface tipo linha de comando. Suponha que os parâmetros que se deseja passar em linha de comando seja [1 2 3 "abd def"]. Para se experimentar esse tipo de passagem de parâmetros com o Dev-C++, usando o ambiente integrado há duas formas.

Uma forma caixa de diálogo “Compile Results”, como mostrado abaixo. Mande essa caixa de diálogo embora por pressionar “Continue”. Obtenha novamente essa caixa de diálogo (talvez por querer mudar os parâmetros) comandando pelo menu Tools - Compile Results (F12). Para que a caixa de diálogo exiba o campo onde se digitam os parâmetros, tecle em “Parameters”, e execute o programa com “>> Execute <<”.



Há ainda outra forma de se fazer a mesma coisa. Comande pelo menu “Options - Environment Options”. Na caixa de diálogo “Environment Options”, preencha o campo apropriado, como mostrado, e não se esqueça de deixar marcado o *checkbox* de “Give the following parameters when executing a compiled project.”



Altere o programa do projeto para que fique como abaixo.

```
// fonte1.cpp
#include <iostream.h>
#include <conio.h> // getche
int main(int argc, char*argv[]) {
    cout << "Hello, world" << endl;
    for (int i=0; i<argc; i++)
        cout << "Arg[" << i << "]: " << argv[i] << endl;
    getche(); // para parar o programa a espera de uma tecla
    return 0;
}
```

Nesse caso, a saída do programa será como abaixo.

```
Hello, world
Arg[0]:C:\user_villas\c\dev_c\p2\p2.exe
Arg[1]:1
Arg[2]:2
Arg[3]:3
Arg[4]:abc def
```

4.4.3 Usando o Help

No Dev-C++, o help é padrão Windows, que é bastante auto explicável. Em especial no help dessa versão é um tutorial sobre

4.4.4 Projetos (programas com múltiplos fontes)

4.5 g++ (do unix)

Um compilador em unix geralmente é chamado de cc (c compiler), ou de gcc (gnu c compilet) ou ainda de g++ (gnu c compiler que implicitamente compila em C++ e não em C). Verifique como é no seu sistema. A menos que se comande de forma diferente, o arquivo executável resultante é a.out. Lembre-se que em unix qualquer arquivo pode ser executável e não apenas os que tem extensão *.exe ou *.com.

Para ver a versão do seu compilador, digite como mostrado abaixo. Uma das últimas versões disponíveis é 2.95.

```
g++ -v
```

Para compilar e linkar

```
g++ main.cpp // programa com 1 só arquivo fonte
g++ main.cpp file1.cpp file2.cpp // programa com 3 arquivos fonte
```

Caso se modifique apenas o main.cpp, pode-se compilar apenas esse arquivo e linkar com os demais com

```
g++ main.cpp file1.o file2.o
```

Ensina-se o compilador a procurar arquivos include em mais um diretório a partir do uso da opção em linha de comando “-I<diretório>”. No Exemplo abaixo acrescenta-se o diretório “include” na busca para includes.

```
G++ -Iinclude main.cpp file1.cpp file2.cpp
```

4.5.1 “Hello world”

Crie o arquivo de texto fonte1.cpp (veja na página 99). Para criar esse arquivo de texto, use um editor de texto, como o vi ou emacs. Compile-o com “g++ f1.cpp”. Execute-o com o comando a.out, que é o arquivo executável final (o nome a.out é padrão, mas nada impede que seja mudado posteriormente).

4.5.1.1 Adicionando argumentos para a linha de comando

Chamar um programa passando argumentos pela linha de comando faz particularmente muito sentido no caso de compiladores que tem interface tipo linha de comando, como é o caso do g++ do unix (que também tem versão para Windows). Ao construir-se um programa, surge um arquivo executável com um nome, digamos “myprog”. Suponha que na linha de comando escrever-mos como abaixo.

```
myprog 1 2 3 "abc def"
```

Suponha também que o programa myprog seja como abaixo

```
#include <iostream.h>
int main(int argc, char**argv) {
    cout << "Hello, world\n";
    int i;
    for (i=0; i<argc; i++)
        cout << "Arg[" << i << "]: " << argv[i] << endl;
    return 0;
}
```

Nesse caso, a saída do programa será como abaixo.

```
Hello, world
Arg[0]:myprog
Arg[1]:1
Arg[2]:2
Arg[3]:3
Arg[4]:abc def
```

4.5.2 Usando o Help

O help do g++ em unix é em grande parte baseado no comando man (manual), que adicionalmente dá ajuda em todo o sistema operacional unix. Por exemplo: para saber ajuda sobre a função c chamada “printf”, na linha de comando digite como abaixo.

```
man printf
```

4.5.3 Projetos (programas com múltiplos fontes)

Para construir programas com g++ a partir de múltiplos fontes, basta adicionar todos os fontes que pertencem ao projeto na linha de comando. Por exemplo: caso um projeto seja composto pelos arquivos f1.cpp, f2.cpp e f3.cpp (sendo que um desses arquivos contém a função main), construa o programa digitando como abaixo na linha de comando.

```
g++ f1.cpp f2.cpp f3.cpp
```

Caso se esteja depurando o f1.cpp, enquanto se mantém constante o conteúdo de f2.cpp e f3.cpp, pode-se compilar apenas o f1.cpp e ligar com f2.o (versão compilada de f2.cpp) e f3.o (idem f3.cpp). Para isso, compile apenas o f2.cpp e f3.cpp com o comando como na linha abaixo.

```
g++ -c f2.cpp f3.cpp
```

Com esse comando, a opção “-c” (compile only) irá fazer gerar o f2.o e f3.o. Para construir o programa executável final a partir da compilação de f1.cpp com f2.o e f3.o, basta comandar como mostrado na linha abaixo.

```
g++ f1.cpp f2.o f3.o
```

Caso exista uma biblioteca estática que se deseje usar num projeto, simplesmente acrescentando-a à linha de comando se está incluindo a biblioteca no projeto. No exemplo abaixo, uma biblioteca chamada “libsomething.a” é acrescentada ao projeto.

```
g++ f1.cpp libsomething.a
```

4.5.4 Bibliotecas

Em unix, as bibliotecas estáticas têm extensão .a e as dinâmicas tem extensão .so (como referência, em DOS/Windows, geralmente a extensão das bibliotecas estáticas é .lib, e das dinâmicas é .dll). No caso de unix, é comum que as bibliotecas dinâmicas coloquem a versão após o .so.

```
unix_static_lib.a  
unix_dynamic_lib.so
```

```
dos_static_lib.lib  
dos_dynamic_lib.dll
```

O utilitário para criar e manter bibliotecas é o “ar”. Para referência completa do “ar”, comande “man ar”.

Para facilitar a compreensão, heis uma sugestão de convenção de nomes para bibliotecas em unix. Pense num “nome” que identifique a biblioteca. A sua biblioteca deve se chamar libnome.a.

4.5.4.1 Incluir uma biblioteca num projeto

Na linha abaixo, compila-se o fonte m1.cpp, que tem chamadas a biblioteca libmylib.a. O arquivo de include usado dentro de m1.cpp está no diretório corrente (.), portanto esse diretório foi incluído na busca de includes.

```
g++ -I. m1.cpp libmylib.a
```

4.5.4.2 Fazer uma biblioteca

Para se criar a biblioteca libmylib.a contento os arquivos myfile1.o, myfile2.o, etc usa-se o comando abaixo:

```
ar -r libmylib.a myfile1.o myfile2.o ...
```

Exemplo: Para se criar a biblioteca libmylib.a com 2 arquivos f1.cpp e f2.cpp.

```
g++ -c f1.cpp f2.cpp // compila os fontes, criando f1.o e f2.o
```



```
ar -r libmylib.a f1.o f2.o // cria library "libmylib.a" contendo f1.o e f2.o
```

4.5.4.3 Examinar uma biblioteca

Para verificar o conteúdo de uma biblioteca estática, use o comando abaixo. Tente esse comando com uma biblioteca que já esteja pronta do seu sistema. Provavelmente há várias no diretório /usr/lib.

```
ar -t unix_static_lib.a
```

```
// exemplo
$ ar -t /usr/lib/libz.a
___.SYMDEF
uncompr.o
gzio.o
crc32.o
compress.o
inflate.o
deflate.o
adler32.o
infblock.o
zutil.o
trees.o
infcodes.o
inftrees.o
inffast.o
infutil.o
```

4.5.5 Fazendo uma biblioteca usando libtool

Libtool é uma ferramenta (gratuita) da gnu para bibliotecas portáteis. Essa ferramenta substitui com vantagens o tradicional ar para gerenciamento de bibliotecas. Veja a referência original sobre libtool em [44]. O Libtool permite linkar e carregar bibliotecas estáticas e também compartilhadas (*shared*), isto é, bibliotecas dinâmicas (o que em Windows corresponde a *dll*). Por usar o libtool para gerenciar as bibliotecas do seu projeto, há que se preocupar apenas com a interface do libtool. Se o projeto faz uso de uma biblioteca com estrutura diferente, o libtool cuida de chamar o compilador e linker para acertar a estrutura da biblioteca. Com isso, a biblioteca binária será instalada de acordo com as convenções da plataforma em questão.

Para usar o libtool, é preciso que essa ferramenta esteja instalada no seu sistema. Caso não esteja, peque-a em [44] e instale-a.

Enquanto uma biblioteca estática é criada a partir de código objeto (*.o), uma biblioteca dinâmica deve ser criada a partir de código independente de posição (*position-independent code* [PIC]). A nível de arquivo, a informação PIC consiste do código objeto normal (*.o), e de um arquivo adicional tipo *library object* (*.lo). Por exemplo: seja os arquivos fonte `plus_1_file.cpp` e `plus_2_file.cpp` abaixo.

```
// plus_1_file.cpp
int plus_1(int i) {
    return i+1;
}
```

```
// plus_2_file.cpp
int plus_2(int i) {
    return i+2;
}
```

Para esses arquivos, pode-se gerar o código independente de posição (plus_1_file.o, plus_1_file.lo, plus_2_file.o e plus_2_file.lo) com o comando abaixo. No exemplo abaixo, usa-se libtool e g++ para gerar os arquivos objeto e independente de posição para plus_1_file.cpp e plus_2_file.cpp.

```
libtool g++ -c plus_1_file.cpp
libtool g++ -c plus_2_file.cpp
```

Uma biblioteca dinâmica precisa de um diretório para se instalada. Um diretório comum para se instalar uma biblioteca dinâmica é /usr/local/lib. No exemplo abaixo, 2 arquivos (plus_1_file e plus_2_file) compõe a biblioteca “libplus_n.la”. A biblioteca é construída no diretório corrente.

```
libtool g++ -o libplus_n.la plus_1_file.lo plus_2_file.lo -rpath /usr/local/lib
```

Seja um programa de teste da biblioteca, como mostrado abaixo.

```
// use_plus_n.cpp
#include <iostream.h>
int plus_1(int);
int plus_2(int);
int main () {
    int k = plus_1(4);
    cout << "k=" << k << endl;
    k = plus_2(3);
    cout << "k=" << k << endl;
    return 0;
}
```

A saída do programa deve ser como mostrado abaixo. Esse programa usa 2 funções globais que estão definidas na biblioteca. Portanto o problema é conseguir linkar o programa.

```
k=5
k=5
```

Para usar a biblioteca diretamente (no caso de a biblioteca não estar instalada no sistema), basta compilar o programa que usa a biblioteca e posteriormente linkar com a biblioteca.

```
g++ -c use_plus_n.cpp
libtool g++ -o test use_plus_n.cpp libplus_n.la
```

Se a biblioteca estiver instalada, digamos no diretório /usr/local/lib, basta indicar o local onde a biblioteca está instalada com a opção -L, como mostrado no exemplo abaixo.

```
g++ -c use_plus_n.cpp
libtool g++ -L/usr/local/lib -o test use_plus_n.o libplus_n.la
```

Nos dois exemplos acima, é gerado o arquivo executável de saída “test”.

4.5.5.1 Instalando uma biblioteca dinâmica

Para se instalar uma biblioteca dinâmica, é preciso usar o aplicativo install, que geralmente encontra-se em /usr/bin/install. Seguindo o exemplo anterior, para que se instale uma biblioteca no sistema, no diretório /usr/local/lib, a partir de um arquivo libplus_n.la já criado, executa-se o comando abaixo.

```
libtool --mode=install /usr/bin/install -c libplus_n.la /usr/local/lib/libplus_n.la
```

Com esse comando, o arquivo libplus_n.la é copiado para /usr/local/lib, e são criados os arquivos libplus_n.so, libplus_n.so.0 e libplus_n.so.0.0.0 também no diretório /usr/local/lib. Isso significa que a versão 0.0.0 da biblioteca está instalada.

Para se gerenciar a versão de uma biblioteca instalada, é preciso em primeiro lugar ser capaz de definir a versão da biblioteca que se pretende instalar.

Exemplo para instalar a vblib. Primeiramente vá para um diretório de trabalho, onde existam os arquivos vblib.cpp e vblib.h (esses arquivos podem ser baixados de www.vbmcgi.org/vblib). Em primeiro lugar, compile todos os fontes (no caso há somente um), com a linha abaixo.

```
libtool g++ -c vblib.cpp
```

O resultado deve ser parecido com o mostrado abaixo.

```
rm -f .libs/vblib.lo
g++ -c vblib.cpp -fPIC -DPIC -o .libs/vblib.lo
g++ -c vblib.cpp -o vblib.o >/dev/null 2>&1
mv -f .libs/vblib.lo vblib.lo
```

Em seguida, crie o arquivo libvblib.la com o comando abaixo

```
libtool g++ -rpath /usr/local/lib -o libvblib.la vblib.lo
```

O resultado deve ser parecido com o mostrado abaixo.

```
rm -fr .libs/libvblib.la .libs/libvblib.* .libs/libvblib.*
gcc -shared vblib.lo -lc -Wl,-soname -Wl,libvblib.so.0 -o .libs/libvblib.so.0.0.0
(cd .libs && rm -f libvblib.so.0 && ln -s libvblib.so.0.0.0 libvblib.so.0)
(cd .libs && rm -f libvblib.so && ln -s libvblib.so.0.0.0 libvblib.so)
ar cru .libs/libvblib.a vblib.o
ranlib .libs/libvblib.a
creating libvblib.la
(cd .libs && rm -f libvblib.la && ln -s ../libvblib.la libvblib.la)
```

Por último, instale a biblioteca dinâmica no diretório adequado (do sistema), no caso /usr/local/lib. Para isso, use o comando abaixo.

```
libtool --mode=install /usr/bin/install -c libvblib.la /usr/local/lib/libvblib.la
```

O resultado deve ser parecido com o mostrado abaixo.

```
/usr/bin/install -c .libs/libvblib.so.0.0.0 /usr/local/lib/libvblib.so.0.0.0
(cd /usr/local/lib && rm -f libvblib.so.0 && ln -s libvblib.so.0.0.0 libvblib.so.0)
(cd /usr/local/lib && rm -f libvblib.so && ln -s libvblib.so.0.0.0 libvblib.so)
/usr/bin/install -c .libs/libvblib.lai /usr/local/lib/libvblib.la
/usr/bin/install -c .libs/libvblib.a /usr/local/lib/libvblib.a
ranlib /usr/local/lib/libvblib.a
chmod 644 /usr/local/lib/libvblib.a
PATH="$PATH:/sbin" ldconfig -n /usr/local/lib
ldconfig: warning: /usr/local/lib/libdb_cxx-3.1.so is not a symlink
ldconfig: warning: /usr/local/lib/libdb-3.1.so is not a symlink
```

```
-----  
Libraries have been installed in:  
/usr/local/lib
```

If you ever happen to want to link against installed libraries in a given directory, LIBDIR, you must either use libtool, and specify the full pathname of the library, or use `~LLIBDIR` flag during linking and do at least one of the following:

- add LIBDIR to the `LD_LIBRARY_PATH` environment variable during execution
- add LIBDIR to the `LD_RUN_PATH` environment variable during linking
- use the `-Wl,--rpath -Wl,LIBDIR` linker flag
- have your system administrator add LIBDIR to `/etc/ld.so.conf`

See any operating system documentation about shared libraries for more information, such as the `ld(1)` and `ld.so(8)` manual pages.

// ainda há o que melhorar !

// Exemplo para usar a plus_n após ter sido instalada:

// Usando a biblioteca sem fazer a sua instalação no sistema.

4.5.6 Debug

Existe uma ferramenta de debug gráfica para unix e sistema GUI tipo X. Mas na prática, eu acabo usando o debug de outro sistema (um bom compilador em Windows) e passo para unix os componentes de programação já testados. Essa é uma estratégia.

Quando é fundamental depurar programas no próprio unix, pode-se usar uma série de artifícios para depurar um programa sem o uso de ferramentas de debug. Por exemplo, pode-se mandar exteriorizar variáveis no meio do programa para verificar o seu conteúdo. Diretivas de compilação podem criar versões de “debug” e “release” do sistema em desenvolvimento, sendo que a versão debug é igual a versão release, a menos do acréscimo de algumas linhas de exteriorização de valores de variáveis.

4.5.7 Definindo um identificador para compilação condicional

A compilação condicional é a compilação de trechos de código fonte a partir de uma condição.

```
// vb100.cpp  
// in unix compile it with the comand line below  
// g++ -DGCC vb100.cpp  
#include <iostream.h>  
int main () {  
    cout << "hello" << endl;  
#ifdef WIN32  
    cout << "I am compiling for WIN32" << endl;  
#endif  
#ifdef GCC
```

```

        cout << "I am using g++ compiler (UNIX)" << endl
            << "and I defined the \"GCC\" constant in the compiler line command " << endl;
#endif
    return 0;
}

```

4.5.8 O pacote RPM do linux

Originalmente, o unix exige uma quantidade de conhecimentos considerável para a instalação de um programa. Geralmente um arquivo compactado é fornecido. Ao ser aberto em um diretório, há que se ler um as instruções num arquivo geralmente nomeado como INSTALL. Essas instruções são algo como executar “./config”, e depois “make install”. Mas na prática, não é incomum que se exija vários parâmetros de configuração a mais na hora da instalação.

O sistema operacional linux possui uma forma de instalação de programas que é bastante mais fácil que a forma tradicional. Essa forma é conhecida como rpm (uma extensão de arquivo). O significado de rpm é Redhat package manager (Redhat é uma distribuição famosa de linux). Procure “rpm” no site do Redhat, ou vá direto para a página deles ensinando a usar rpm, em [43]. O resumo de rpm é mostrado nessa seção.

O pacote rpm é compatível com diversos variantes de unix, e não apenas para linux.

Dentre os serviços que o rpm oferece incluem-se

- Instalar ou dar upgrade do software, verificando dependências.
- Enquanto instala o software, verificar se o executável fica pronto para usar.
- Recuperar arquivos que por ventura tenham sido acidentalmente apagados de uma instalação prévia.
- Informar se o pacote já está instalado.
- Descobrir a qual pacote corresponde um determinado arquivo.

4.5.8.1 rpm binário e rpm com fonte

Há dois tipos de arquivos rpm. Um deles é binário (pré-compilado), e o outro vem com os fontes, e instala-se a partir da compilação (forma mais lenta de instalar, contudo mais segura e mais portátil).

A designação do rpm binário é *.rpm. A designação do rpm com fonte é *.src.rpm.

4.5.8.2 Alguns comandos do rpm

Veja todos os comandos do rpm rodando simplesmente

rpm

- Instalando um pacote rpm:

`rpm -i <package>.rpm`

- Desinstalando um pacote rpm:

`rpm -e <package_name>`

- Instalando um rpm diretamente da rede (sem haver cópia local).

`rpm -i ftp://<site>/<directory>/<package>.rpm`

Exemplo:

`rpm -i ftp://ftp.redhat.com/pub/redhat/rh-2.0-beta/RPMS/foobar-1.0-1.i386.rpm`

- Verificando o sistema (conferindo o que está instalado)

`rpm -Va`

- Digamos que você encontra um arquivo e quer saber de que pacote rpm ele pertence. No exemplo abaixo, o arquivo “/home/httpd/icons/uuencoded.gif” está sendo.

`rpm -qf /home/httpd/icons/uuencoded.gif`

A saída deve ser como abaixo

`apache-1.3.12-2`

- Pegando informações de um rpm sem instala-lo

`rpm -qpi <package>.rpm`

- Verificando quais

`rpm -qpl <package>.rpm`

4.5.8.3 Construindo um rpm

4.5.8.3.1 Introdução

Construir um pacote rpm é relativamente fácil de se fazer, especialmente se você tem o fonte do software para o qual quer fazer o pacote. O procedimento passo a passo é mostrado abaixo.

1. Prepare o código fonte do pacote, deixando-o pronto para compilar.
2. Faça um arquivo de especificação (*spec file*) para o pacote.
3. Execute o comando de construir (*build*) o pacote rpm.

O arquivo de especificação é um arquivo texto requerido para se construir um pacote rpm. Nesse arquivo texto há uma descrição do pacote, instruções para permitir contruí-lo, e uma lista de arquivos binários que são instalados com o pacote. O melhor é seguir a convenção do arquivo de especificação, como mostrado no exemplo abaixo. Com essa convenção, o sistema poderá lidar com múltiplas versões de um pacote com o mesmo nome. Recomendo que a descrição do software, nome, etc. seja feita no idioma inglês.

Summary: A program that ejects removable media using software control.

Name: eject

Version: 2.0.2

Release: 3

Copyright: GPL

Group: System Environment/Base

Source: <http://metalab.unc.edu/pub/Linux/utils/disk-management/eject-2.0.2.tar.gz>

```

Patch: eject-2.0.2-buildroot.patch
BuildRoot: /var/tmp/%{name}-buildroot

%description
The eject program allows the user to eject removable media
(typically CD-ROMs, floppy disks or Iomega Jaz or Zip disks)
using software control. Eject can also control some multi-
disk CD changers and even some devices' auto-eject features.

Install eject if you'd like to eject removable media using
software control.

%prep
%setup -q
%patch -p1 -b .buildroot

%build
make RPM_OPT_FLAGS="$RPM_OPT_FLAGS"

%install
rm -rf $RPM_BUILD_ROOT
mkdir -p $RPM_BUILD_ROOT/usr/bin
mkdir -p $RPM_BUILD_ROOT/usr/man/man1

install -s -m 755 eject $RPM_BUILD_ROOT/usr/bin/eject
install -m 644 eject.1 $RPM_BUILD_ROOT/usr/man/man1/eject.1
Prep
%clean
rm -rf $RPM_BUILD_Preproot

%files
%defattr(-,root,root)
%doc README TODO COPYING ChangeLog

/usr/bin/eject
/usr/man/man1/eject.1

%changelog
* Sun Mar 21 1999 Cristian Gafton <gafton@redhat.com>
- auto rebuild in the new build environment (release 3)

* Wed Feb 24 1999 Preston Brown <pbrown@redhat.com>
- Injected new description and group.

```

4.5.8.3.2 O header

Explicação item a item:

- Summary: explicação em uma linha sobre o propósito do pacote
- Name: um identificador para o pacote
- Version: a versão do pacote
- Release: é como uma sub-versão
- Copyright: diga como quer a licença do pacote. Você deve usar algo como GPL, BSD, MIT, public domain, distributable ou commercial.

- **Group:** o grupo ao qual o pacote pertence num nível do instalador RedHat.
- **Source:** essa linha aponta para o diretório raiz do programa fonte. É usado para o caso de você precisar pegar o fonte novamente e checar por novas versões. É importante lembrar que o nome do arquivo dessa linha precisa ser exatamente igual ao que você tem de fato no sistema (ou seja, não mude o nome dos arquivos fonte dos quais fez download). Os arquivos fonte estão geralmente compactados com tar e depois com gzip, ou seja, tem a extensão *.tar.gz. Você pode especificar mais de um arquivo fonte, como mostrado no exemplo abaixo.

```
source0: mySource.tar.gz
source1: myOtherSource.tar.gz
source2: myYetOtherSource.tar.gz
```

Esses arquivos vão para o diretório SOURCES. (A estrutura de diretórios do pacote rpm é mostrada mais abaixo.)

- **Patch:** Esse é o local onde você encontra os arquivos que “concertam” uma versão já disponibilizada (esse procedimento é chamado de *patch*). Da mesma forma que para os arquivos fonte, o nome dos arquivos precisa ser exatamente igual ao original, e pode haver mais de um. O exemplo abaixo ilustra-o. Os arquivos de patch serão copiados para o diretório SOURCES.

```
Patch0: myPatch.tar.gz
Patch1: myOtherPatch.tar.gz
Patch2: myYetOtherPatch.tar.gz
```

- **Group:** Essa linha é usada para dizer para programas instaladores de alto nível (tal como gno rpm) onde salvar o programa do pacote e sua estrutura hierárquica. Você pode encontrar ...
- **BuildRoot:** É uma linha que permite especificar um diretório como “root” para construir e instalar o pacote. Você pode usar essa opção para ajudar a testar seu pacote antes de realmente instalar na máquina.
- **%description:** É onde se escreve a descrição do pacote. Recomenda-se enfaticamente o uso do idioma inglês para a descrição. Nesse campo pode-se usar várias linhas, como mostrado no exemplo.

4.5.8.3.3 Preparação (prep)

É a segunda seção do arquivo de especificação (*spec file*). É usado para obter parâmetros para construir o pacote. O que a seção prep faz realmente é executar scripts no shell. Você pode simplesmente fazer um script seu para construir o pacote, e colocar após o tag %prep. Contudo, há alguns macros que podem ser úteis, e são mostrados abaixo.

- **-n name.** Esse macro define o nome do diretório para construir o pacote com o nome indicado. O nome implícito (*default*) é \$NAME\$VERSION.

Outras possibilidades são \$NAME, ou \${NAME}\${VERSION}, ou o que for usado nos arquivos tar. Note que o uso da letra “\$” *não* são variáveis especificadas no spec file. Estão apenas usadas aqui no lugar de um nome de exemplo. Você precisa usar um nome real no seu pacote, e não uma variável.

- -c. Essa opção irá fazer mudar para o diretório de trabalho *antes* de executar a descompactação com untar.
- -b. Essa opção irá descompactar os fontes antes de mudar de diretório. É útil apenas quando se usa múltiplos arquivos fonte.
- -a. Essa opção irá descompactar os fontes após mudar para o diretório.
- -T. Essa opção sobre-escreve a ação implícita de descompactar com untar os fontes, e requer o uso da opção -a ou -b para obter os principais fontes descompactados. Você precisa disso quando são usados fontes secundários.
- -D. Use essa opção para *não* apagar o diretório após desempacota-lo.

// ainda há o que melhorar

5 Princípios de C/C++

5.1 O primeiro programa

```
#include <stdio.h>
int main()
{
    printf("Hello world\n");
    return 0;
}
```

A saída do programa é:

```
Hello world
```

Basicamente, um compilador é uma ferramenta que transforma um arquivo de texto (conhecido como texto fonte) num arquivo executável. Chama-se esse procedimento “construir (*build*) um executável”. “Construir” é o procedimento que inclui “compilar” e “ligar”, como será explicado em detalhe mais tarde. As seções abaixo abordam os conceitos básicos sobre como se pode escrever o arquivo de texto fonte.

5.2 Formato livre

Formato livre (*free format*) que significa que o compilador observa o texto fonte sem atenção a posicionamento relativo dos caracteres. Em FORTRAN, que por exemplo não possui formato livre, o código fonte só pode ser escrito a partir da coluna 7 do texto e o return no final da linha significa que o comando acabou. A maioria das linguagens modernas como C e C++ podem ser escritas em formato livre o que significa também que no lugar de 1 espaço podem haver qualquer número de espaços, returns e tabs sem qualquer alteração na compilação.

Com o formato livre, as funções `myinc` e `myinc2` abaixo são exatamente iguais. Apesar de o formato ser livre, há regras recomendadas de indentação, que são definidas com finalidade de facilitar a legibilidade do programa.

```
int myinc(int in) {
    return in+1;
}

int
myinc2
(
int
in
)
{
return
in
+
}
```

```
1
;
}
```

5.3 Chamada de função

A chamada de uma função corresponde à chamada de uma sub-rotina, como ilustrado na figura abaixo.

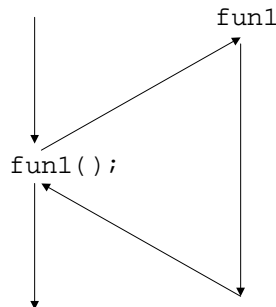


Figura 6: representação gráfica do fluxo de um programa na chamada de uma função

Para se chamar uma função em C, basta escrever o nome da função seguido de (). Caso haja parâmetros, eles devem ser colocados dentro dos parênteses, separados por vírgulas. Mesmo que não haja parâmetros é necessário que se escreva o abre-e-fecha parêntesis vazio (). Depois da chamada da função, como depois de qualquer comando, é necessário que se pontue com ponto-e-vírgula ;.

Caso haja retorno da função, uma variável de retorno colocada antes da função e o uso do operador = fazem com que a variável receba o retorno da função. Não é necessário que o retorno de uma função seja usado. Caso não seja necessário usar o retorno de uma função, basta ignorar esse valor de retorno.

Exemplo:

```
double triple(double z)
{
    return 3*z;
}
int main()
{
    double d = 2.2;
    double dd = triple(d);
    triple(d); // o retorno não é usado. Nesse caso não faz nada, mas é legal.
    return 0;
}
```

5.4 Declaração e definição de funções

É importante entender bem a diferença desses dois conceitos.

1. “Declarar uma função” significa dizer qual é o protótipo dessa função, isto é, quantos são os parâmetros e de que tipo, bem como o tipo de retorno.
2. “Definir uma função” significa escrever o conteúdo da função. Ao se definir uma função, se está declarando também. Mas não o contrário.

O compilador somente pode gerar código a partir de identificadores que tenham sido previamente declarados.

Declaração	Definição
<code>int plus_3(int);</code>	<code>int plus_3(int in) { return in+3; }</code>

5.5 Comentários

É muito útil que se possa inserir comentários num texto fonte. Em C, os comentários são tudo que se situa entre a abertura de comentário `/*` e o fechamento de comentário `*/`. Veja o exemplo abaixo.

```
int num_alunos; /* variável que armazena o numero de alunos */
num_alunos = 4; /* carrega o número de alunos que há na primeira fila */
/* etc */
```

Uma das utilidades de um comentário é que se retire temporariamente do programa um trecho de código. Chama-se isso de “comentar para fora”¹⁰. Mas pela definição da linguagem, os comentários não aninham¹¹ (embora em algumas implementações do compilador os comentários aninhem). Portanto uma tentativa de se comentar para fora um trecho de programa que contenha comentários pode levar a problemas. Veja o exemplo abaixo. Nesse caso, o início do comentário para fora é fechado no fim da segunda linha. Portanto o código `num_alunos=4;` *será* compilado.

```
/* ===== comentado para fora - início
int num_alunos; /* variável que armazena o numero de alunos */
num_alunos = 4; /* carrega o número de alunos que há na primeira fila */
===== comentado para fora - fim */
/* etc */
```

Em C++, além de se usar o comentário como em C, usa-se também outro tipo de comentário. C++ considera comentário tudo que estiver a direita de `//`, sendo desnecessário fechar o comentário, pois o fim da linha já significa fim de comentário. Veja o exemplo abaixo.

```
int num_alunos; // variável que armazena o numero de alunos
num_alunos = 4; // carrega o número de alunos que há na primeira fila
// etc
```

5.6 Identificador

Identificador (*identifier*) se refere as palavras que o programador coloca no programa para dar nome a suas variáveis, funções, tipos, etc. Em C e C++, que são linguagens fortemente tipadas, todos identificadores devem ser declarados antes de serem usados, conforme será visto.

¹⁰ Pode-se também tirar temporariamente trechos de código com diretivas de compilação.

¹¹ Aninhar em inglês é *nest*

```

void g()
{
    int i;
    // int é palavra reservada; "i" é um identificador que representa uma
    // ocorrência (instância) de uma variável do tipo int (inteiro).
    i = 3;
    // "i" é um identificador que representa uma variável tipo int, declarada acima.
    // essa variável recebe uma constante inteira 3.
    float area_do_circulo(float raio);
    // float é palavra reservada. Essa linha de programa é a declaração de uma
    // função, cujo identificador é "area_do_circulo". Essa função
    // recebe um parâmetro que é uma variável tipo float identificador é "raio".
}

```

5.7 Constantes literais

Pode-se carregar valores constantes literais em variáveis, conforme mostrado no exemplo abaixo.

```

int    i  = 3;
float  f  = 4.5;
double pi = 3.14;
char   ch = 'c';
char *string = "Bom dia"; // uma constante string retorna um ponteiro para char

```

5.8 Escopo

O escopo significa “mira, alvo, intenção”¹². Para a programação em C++, significa um trecho de código onde é permitido que novas definições tomem precedência sobre definições feitas anteriormente. Esse trecho de código é delimitado por { (abre escopo) e } (fecha escopo).

Uma variável declarada fora de qualquer função é uma variável global. Qualquer função é definida dentro de um escopo. As variáveis que são parâmetros da função, bem como as variáveis declaradas dentro da função existem apenas enquanto existir o escopo da função. As definições dentro da função tem precedência sobre declarações globais.

```

int i; // variável global, definida fora de uma função
void f1()
{
    int i; // variável no escopo da função f1, diferente da variável global
    i = 3; // a constante 3 é carregada na variável local da função f1
}
void f2()
{
    i = 5; // não há variável local chamada "i", portanto a constante 5
    // será armazenada na variável global "i".
}

```

Outro exemplo:

```

void f1()
{
    int i; // declaro a variável "i"
    i = 3; // atribuo uma constante a essa variável
}

```

¹² Segundo o dicionário Aurélio

```

{ // abro um escopo interno
    int i; // as variáveis definidas no escopo interno não são as mesmas
          // que aquelas fora do escopo; a declaração interna toma precedência

    i = 5; // atribuo uma constante a variável interna
} // fecho o escopo que foi aberto

// o que aconteceu dentro do escopo interno não interessa fora dele
// aqui fora, "i" continua a valer 3;
}

```

5.9 Tipos de dados padrão (Standard Data Types)

Data Type	n de bytes	range
char	1	-128..127
unsigned char	1	0..255
int	2	-32768..32767
unsigned int	2	0..65535
long	4	$-2^{31}..2^{31}-1$
unsigned long	4	$0..2^{32}$
float	4	
double	8	
long double	10	

Declaração:

```

<data_type> <instance_user_identifier_list>;           // em inglês
ou
<tipo_de_dados> <lista_de_ocorrências_de_identificadores_do_usuario>; // em português

```

Exemplo:

```

int i;           // i é ocorrência de variável tipo int
double d1,d2,d3; // 3 ocorrências de variáveis tipo double

```

5.10 Palavras reservadas do C++ (keywords)

As palavras reservadas do C++ são um conjunto de identificadores que são previamente definidas antes de começar a compilação.

Palavras reservadas de C++					
asm	continue	float	new	signed	try
auto	default	for	operator	sizeof	typedef
break	delete	friend	private	static	union
case	do	goto	protected	struct	unsigned
catch	double	if	public	switch	virtual
char	else	inline	register	template	void
class	enum	int	return	this	volatile
const	extern	long	short	throw	while

É proibido usar uma palavra reservada como um identificador definido pelo usuário. Por exemplo, é proibido que se crie uma função como abaixo.

```
void continue() // ERRO ! o nome da função é uma palavra reservada
{
    printf("continue"); // isso não é um erro, pois a palavra reservada
    // está dentro de uma constante string, ou seja,
    // é uma constante literal
}
```

Além das palavras reservadas da linguagem, há uma lista de identificadores que estão definidos na biblioteca padrão de C / C++, por exemplo “printf”. Não é recomendável que se use um identificador que pertença a essa lista. É proibido que se declare 2 identificadores com significados diferentes. Caso se use um identificador que também existe na biblioteca padrão, poderá haver conflito entre as duas declarações e isso gera um erro de compilação.

5.11 Letras usadas em pontuação

!	%	^	&	*	()	-	+	=	{
}		~	[]	;	'	:	"	<	>
?	,	.	/							

5.12 Letras usadas em operadores

->	++	--	.*	->*	<<	>>	<=	>=	==	!=
&&		*=	/=	%=	+=	-=	<<=	>>=	&=	^=
=	::									

5.13 Exercícios resolvidos

- 1) Diga quais dos arquivos *.cpp abaixo compilam, e quais não compilam. Dos que não compilam, altere-os para que compilem.

```
// a.cpp
void fun_2()
{
    int i = fun();
}
//-----
// b.cpp
int fun();
void fun_2()
{
    int i = fun();
}
//-----
// c.h
double f(int i);
//-----
// d.cpp
#include "c.h"
double f2(int k);
    return f(k + 5);
}
//-----
// e.cpp
void f()
```

```

{
    double volatile = 1.1;
}
//-----
// f.cpp
float f(float z)
{
    return z + 2.2;
}
void f2()
{
    f(3.3);
    4;
}

```

5.13.1 Resposta

O arquivo 1.a) não compila, pois se está gerando código dentro da função “fun_2”, mas o identificador “fun” não foi declarado. Uma alteração possível para esse arquivo é o arquivo 1.b), que compila.

O arquivo 1.d) compila. A inclusão de c.h permite a declaração da função “f”, que retorna double e tem um parâmetro, que é um int. No arquivo d.cpp, está definida a função “f2”, que tem o mesmo protótipo da função “f”, e que chama essa mesma função (isso é, gera código a partir do identificador “f”).

O arquivo 1.e) não compila, pois “volatile” é palavra reservada e não pode ser usado como identificador.

O arquivo 1.f) compila, mas o programa em questão é um tanto esquisito. Em C/C++, tudo retorna, mesmo que o retorno não seja aproveitado. Caso o retorno não seja aproveitado, nada acontece, o que se escreveu não é um erro, e o programa compila. A função “f” recebe um parâmetro float, e retorna um float. A função f2 chama a função f, mas o retorno não é usado, e isso não é um erro. Abaixo, a constante literal “4” é “chamada”. Com isso, a linha retorna “4”, e nada acontece, mas igualmente a linha não é um erro.

6 Estrutura do Compilador

6.1 Entendendo o Compilador

Caso se utilize o sistema operacional Windows [unix], o que o compilador fez no exemplo “hello world” foi compilar o seu texto fonte (*.c ou *.cpp) criando um código compilado (*.obj [*.o]). Em seguida foi chamado o linker que ligou este último arquivo com a biblioteca padrão (*standard library*) do C / C++ (c.lib) e criou um arquivo executável (*.exe [qualquer extensão]). Um compilador comercial (como Visual C++, ou Borland C++ Builder) possui geralmente ambiente integrado que chama a execução de (*.exe) sob debug, mas como não foi ativada nenhuma função de debug, nada além da execução normal ocorreu e o programa rapidamente chega ao fim, retornando então ao ambiente integrado.

Um programa fonte passa por um processo de construção (*build*), para que seja produzido o programa executável. O processo de construção do programa é separado em duas etapas – compilação e ligação, como será explicado com mais detalhes. A figura 7 ilustra o processo de construção de um programa.

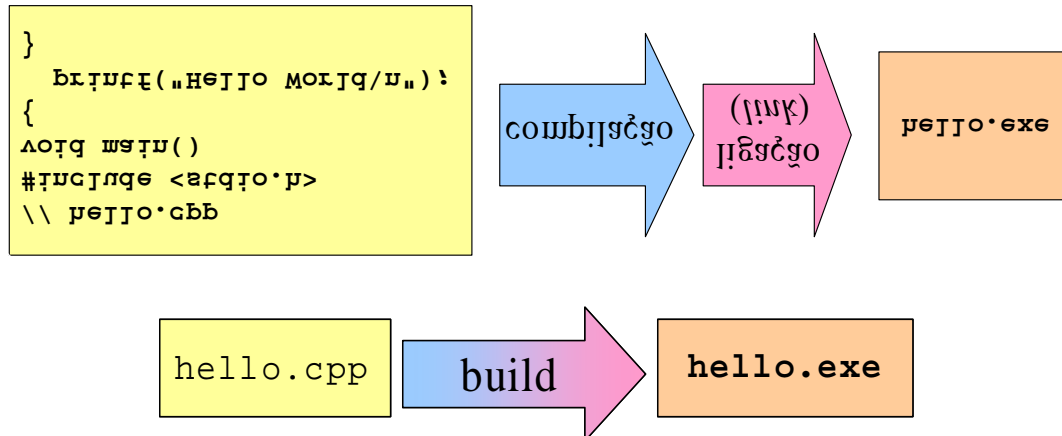


Figura 7: Processo de construção de um programa executável

Os programas em C++ são escritos como um conjunto de funções. Uma função em C++ é uma subrotina que retorna um valor do tipo *return_type*, possui um nome identificador *function_name* e possui um número maior ou igual a zero de argumentos (mesmo que não haja argumento nenhum, é necessário o abre e fecha parênteses “ () ”). O corpo da função é escrito no *block statement*, que é o espaço entre as chaves “ { } ”. Os comentários (não geram código compilado) são o que estiver entre “ /* ” e “ */ ” ou a direita de “ // ”.

```
<return_type> function_identifier (<arguments>)  
{  
    // function body
```

```
}
```

Ou em português.

```
<tipo_de_retorno> identificador_da_função (<argumentos>)  
{  
    // corpo da função  
}
```

Em C++, o programa é a execução da função `main`¹³. A função `main`, bem como todas as funções, pode chamar quaisquer funções definidas no universo do programa. O arquivo `hello.cpp` possui apenas a função `main`. Como não se deseja retorno desta função, escreve-se a palavra reservada `void` (cancelado) como o tipo de retorno da função `main`. O corpo da função contém apenas a chamada da função `printf`, da biblioteca padrão, cuja finalidade é colocar no console de saída (tela) o argumento que recebeu, que é a string `"hello world\n"`. O `"\n"` no final significa `<return>`, isto é, mudar de linha. A linha `"#include <stdio.h>"` no início do programa serve para declarar o protótipo (*prototype*) da função `"printf"`, conforme será visto mais tarde.

6.2 Protótipos (*prototypes*)

Os identificadores precisam ser declarados antes de serem usados¹⁴, ou seja, antes que se possa gerar código a partir deles. Caso o identificador seja o nome de uma função global, essa função pode ser definida antes de ser usada, ou pode ser apenas declarada (protótipo) antes de ser usada. No segundo caso, é preciso que a definição exista em algum lugar do projeto, pois do contrário ocorre erro de ligação (*link*). A definição de uma função inclui a declaração dessa função, mas não vice-versa.

Para declarar um identificador como função deve-se declarar o protótipo (*prototype*) da função. O protótipo da função contém apenas o tipo do retorno, o identificador da função, os argumentos entre parêntesis e um ponto-e-vírgula (;) no final. O protótipo não contém o corpo propriamente dito da função nem gera código executável, **apenas faz a sua declaração** para permitir que o programa abaixo possa usar a função.

Seja o programa abaixo. Nesse programa, define-se (e portanto declara-se) a função global `"hello"`, e em seguida a função global `"bye"`. Por último, define-se a função `main`, que chama a função `hello` e a função `bye`. O programa está correto (pode ser construído sem erros), pois quando se está gerando código a partir dos identificadores das funções `hello` e `bye`, esses identificadores já estão definidos.

```
#include <iostream.h>  
// definição da função hello (inclui declaração)  
void hello()
```

¹³ `"main"` significa `"principal"`.

¹⁴ Em C não é obrigatório declarar os protótipos das funções antes de usá-las, é apenas fortemente recomendável. Em C++ é obrigatório.

```

{
    cout << "Hello, world" << endl;
}

// definição da função bye (inclui declaração)
void bye()
{
    cout << "Bye bye, world" << endl;
}

// definição da função main
int main()
{
    hello(); // chama-se a função hello
    bye();   // chama-se a função bye
    return 0;
}

```

Saída do programa

```

Hello, world
Bye bye, world

```

Considere agora uma variante desse programa, como mostrado abaixo. Nessa variante, primeiro declara-se os protótipos das funções globais hello e bye. A declaração desses protótipos não gera código, mas faz o compilador aprender o significado dos identificadores. Com isso, quando o compilador pode gerar código normalmente na função main (que chama hello e bye). Posteriormente à função main está a definição das funções hello e bye, em compatibilidade com os protótipos previamente declarados. Esse programa é construído sem problemas, e dá o mesmo resultado do programa anterior.

```

#include <iostream.h>

// declaração de funções globais (não gera-se código)
void hello(); // declaração (protótipo) da função hello
void bye();   // declaração (protótipo) da função hello

// definição da função main
int main()
{
    hello(); // chama-se a função hello
    bye();   // chama-se a função bye
    return 0;
}

// definição da função hello
void hello()
{
    cout << "Hello, world" << endl;
}

// definição da função bye
void bye()
{
    cout << "Bye bye, world" << endl;
}

```

Como exercício, imagine que o programa acima fosse modificado e as linhas após a função main fossem apagadas, de forma que a definição das funções hello e bye

não existisse mais. Nesse caso, o programa compilaria, mas não passaria pela ligação (*link*), que acusaria a falta das funções que foram apagadas.

6.3 Projetos em C/C++

Um projeto em C/C++ é a forma de construir um programa executável a partir de mais de um arquivo fonte (*.c ou *.cpp). A forma como se representa a informação de quais arquivos fonte fazem parte de um dado projeto não é padrão. Ou seja, compiladores diferentes tem formas diferentes de informar quais arquivos fonte fazem parte de um projeto (ainda que usem os mesmos arquivos fonte).

Para construir o programa executável a partir de um projeto, o compilador compila cada um dos arquivos fonte (*.c ou *.cpp), gerando arquivos objeto de mesmo nome e extensão *.obj (no caso de Windows) ou *.o (no caso de unix). Cada arquivo objeto contém a informação do código compilado de funções globais (e outras) definidas no arquivo fonte correspondente.

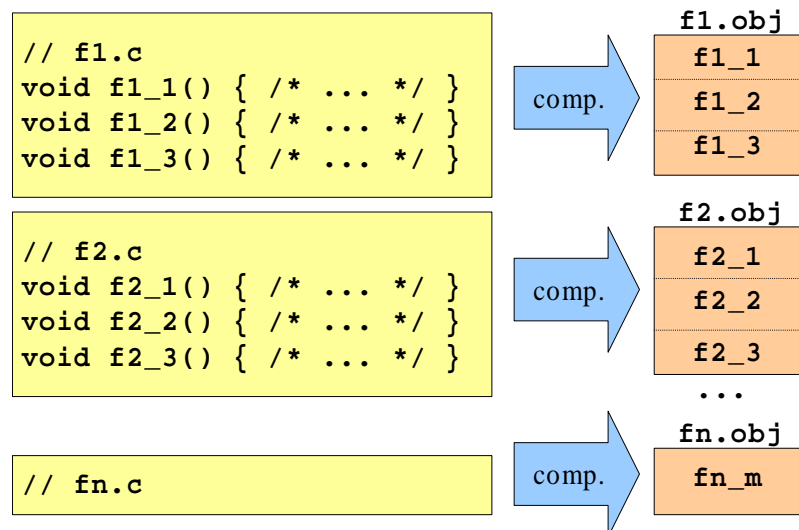


Figura 8: Etapa de compilação de um projeto em C/C++

Após concluída a etapa de compilação de todos os arquivos fonte, ocorre a etapa de ligação. Nessa etapa, todos os arquivos objeto do projeto são incluídos, e também a biblioteca padrão (*standard library*). A biblioteca padrão é automaticamente incluída em todos os projetos. Um e somente um arquivo fonte deve conter a função `main` (do contrário, ocorre um erro de *link*). O programa executável gerado geralmente tem o nome do projeto.

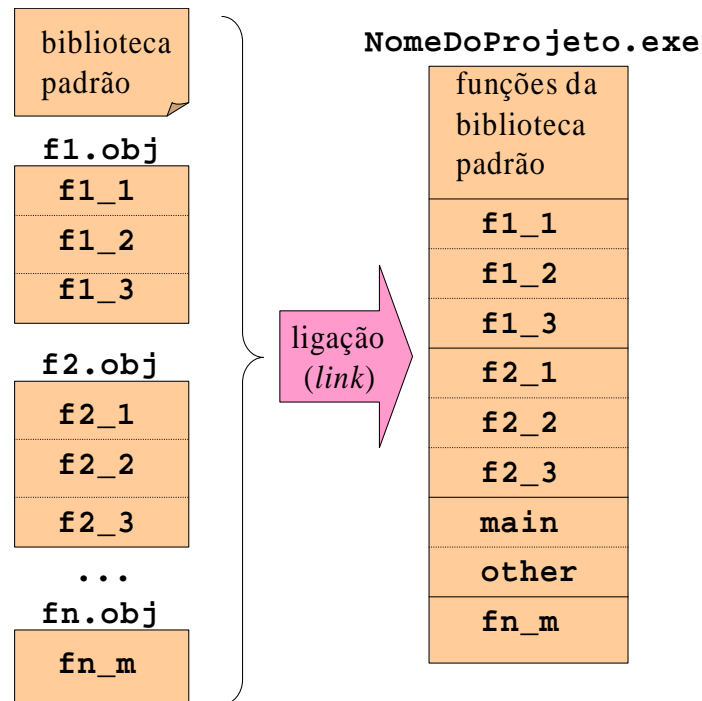


Figura 9: Etapa de ligação de um projeto em C/C++

Funções do programa compilador

- Cada arquivo fonte *.c (*.cpp) é compilado gerando *.obj (ou *.o).
- Cada arquivo fonte *.c (*.cpp) é compilado como se outros arquivos do projeto não existissem.
- No início da compilação de cada arquivo *.c (*.cpp), o compilador apenas conhece o vocabulário das palavras reservadas, e a gramática da linguagem C/C++.
- O compilador deve conseguir compilar o código fonte numa única passagem, ou haverá erro de compilação.
- Ao analisar um trecho de código fonte, o compilador faz uma das 2 coisas:
 - Gera código.
 - Aprende o significado de novos identificadores, e com isso passa a poder compilar com o que aprendeu.
- O compilador não pode gerar código a partir de identificadores que não tenham sido previamente declarados. A tentativa de fazê-lo gera erro de compilação.
- Um mesmo arquivo fonte *.c (*.cpp) pode pertencer a mais de um projeto.

6.4 Header Files (*.h)

Agora pode-se explicar melhor a necessidade da linha `#include <iostream.h>` no início de vários arquivos fonte. Esta linha faz incluir um arquivo chamado `iostream.h`, que é um dos *header files* padrão (sempre fornecidos com o compilador C++). O arquivo `iostream.h` é um arquivo de texto (pode ser examinado com o editor) em que está escrito (entre outras coisas) o protótipo da função `printf` que foi usada no programa de exemplo. Desta forma, quando o compilador encontra a chamada da função `printf`, já sabe o seu protótipo e pode conferir se a chamada da função foi feita corretamente.

Os arquivos com a extensão `.h` são chamados de *header* e sua finalidade é de serem incluídos nos programas fonte com a diretiva `#include <*.h>`, para includes padrão do compilador e `#include "*.h"` para includes que o programador desenvolver. Os arquivos de *header* geralmente contém, além de protótipos de funções, definições de tipos, macros (`#define`), etc. Em C os *headers* não devem conter trechos que gerem código executável como o corpo de funções, já em C++ podem conter códigos curtos inline, conforme será visto mais tarde.

Os arquivos header outro tipo de arquivo fonte. Esses arquivos são feitos para serem incluídos em arquivos `*.c`, `*.cpp` ou mesmo outro arquivo `*.h`. A inclusão é feita com a diretiva `#include <nome.h>`. O compilador substitui essa diretiva pelo conteúdo do arquivo `nome.h` e segue a compilação. Os headers do sistema são incluídos com o nome entre `< >`, enquanto os headers do programador são incluídos com `#include "nome.h"`.

Nesses arquivos geralmente são escritas declarações úteis para permitir a compilação do código que vem a seguir. No exemplo abaixo, há um programa que foi gerado a partir de 2 arquivos fonte chamados `plus_1_file.cpp` e `main.cpp`. Para que se possa compilar o arquivo `main.cpp` é necessário que se declare o identificador `plus_1`, e isso é feito no arquivo `myheader.h`.

```
// myheader.h
int plus_1(int in); // protótipo de uma função
```

```
// plus_1_file.cpp
int plus__1(int in)
{
    return in + 1;
}
```

```
// main.cpp
#include <iostream.h>
#include "myheader.h"
int main()
{
    int i=2;
    cout << "i+1=" << plus_1(i) << endl;
    return 0;
}
```

}

6.5 Biblioteca (*library*)

Um arquivo executável (também chamado de “aplicativo”) é um produto para o usuário. Analogamente, uma biblioteca é um produto para o programador. Uma biblioteca não é executável. É um conjunto de funções compiladas (*.obj) mas ainda não ligadas a nenhum arquivo executável. Para se criar uma biblioteca, todo compilador possui uma ferramenta de criação de bibliotecas.

Todo compilador possui uma biblioteca padrão (*standard library*), que é uma biblioteca como outra qualquer. Em alguns compiladores os arquivos fonte para a criação da biblioteca padrão estão disponíveis. A única diferença da biblioteca padrão e outras bibliotecas é o fato de que a biblioteca padrão por definição está sempre disponível em qualquer compilador. Qualquer outra biblioteca que se use deve ser adicionada manualmente. Por isso, geralmente quem usa uma biblioteca não padrão “x”, costuma ter (ou querer ter) os arquivos fonte dessa biblioteca. Assim, pode-se criar a biblioteca sempre que necessário, em qualquer compilador.

6.5.1 Utilizando Bibliotecas prontas

Uma biblioteca é um conjunto de funções prontas para serem usadas. Geralmente uma biblioteca se apresenta como 2 arquivos: nome_da_biblioteca.lib

1. nome_da_biblioteca.lib
2. nome_da_biblioteca.h

Para se usar uma biblioteca feita por outra pessoa, basta acrescentar a biblioteca no projeto. Um projeto de teste típico com uma biblioteca conterá 2 arquivos

1. nome_da_biblioteca.lib
2. main.cpp (um arquivo fonte com a função main)

Geralmente a biblioteca vem com pelo menos 1 arquivo header, com as declarações das funções que contém a biblioteca. Portanto um programa que use as funções da biblioteca deverá incluir o tal header.

Exemplo:

```
#include "nome_da_biblioteca.h"
int main ()
{
    fun_da_biblioteca();
    return 0;
}
```

6.5.2 Fazendo bibliotecas

Além de se criar arquivos executáveis, o compilador pode também criar uma biblioteca. Se um executável (um aplicativo) é um produto para o usuário, uma biblioteca deve ser encarada como um produto para o programador. Uma biblioteca não é executável, é um conjunto de funções compiladas (*.obj) mas ainda não ligadas a nenhum arquivo executável. Para se criar uma biblioteca, todo compilador possui uma ferramenta de criação de bibliotecas.

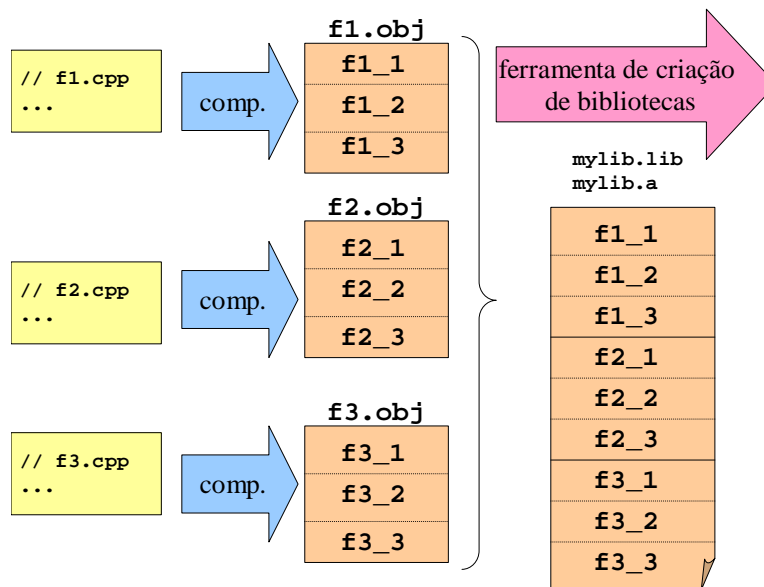


Figura 10: diagrama da criação de bibliotecas

O objetivo dessa prática é entender a criação e uso de bibliotecas. Nesse sentido, será feita uma biblioteca que não tem utilidade prática. Isto é, as funções da biblioteca não são úteis realmente; são apenas funções bastante simples para teste.

Prática: Fazer uma biblioteca com as especificações abaixo:

1. Nome da biblioteca: `plus_n`, contendo 5 funções: `plus_1`, `plus_2`, `plus_3`, `plus_4`, `plus_5`. Cada uma dessas funções deve somar 1, 2, 3, 4 ou 5 a entrada (conforme o nome), e retornar esse valor.

Resposta: Para a implementação das 5 funções serão criados 5 arquivos diferentes (não é necessário que os arquivos sejam diferentes, mas é recomendável para que a biblioteca seja mais “fatiada”, isto é, que cada fatia da biblioteca entre ou deixe de entrar no programa executável final de acordo com a necessidade do linker). Os 5 arquivos `plus_?.cpp` estão mostrados abaixo.

```
// plus_1.cpp
int plus_1 (int in) { return in+1; }
```

```
// plus_2.cpp
```



```
int plus_2 (int in) { return in+2; }
```

```
// plus_3.cpp  
int plus_3 (int in) { return in+3; }
```

```
// plus_4.cpp  
int plus_4 (int in) { return in+4; }
```

```
// plus_5.cpp  
int plus_5 (int in) { return in+5; }
```

Para permitir o uso da biblioteca, é preciso que haja um arquivo de header com os protótipos das funções que existem. O arquivo merece o mesmo nome da biblioteca, com a extensão .h. O arquivo plus_n.h é mostrado abaixo.

```
// plus_n.h  
int plus_1 (int in);  
int plus_2 (int in);  
int plus_3 (int in);  
int plus_4 (int in);  
int plus_5 (int in);
```

Utilizando-se a ferramenta de criação de bibliotecas do seu compilador, a partir dos arquivos fonte mostrados acima, cria-se a biblioteca plus_n.lib (ou libplus_n.a). Para se testar a biblioteca, é preciso que exista um programa principal com a função main (a biblioteca não contém a função main). O programa abaixo test_plus_n.cpp usa funções da biblioteca.

```
// test_plus_n.cpp  
#include <iostream.h>  
#include "plus_n.h"  
int main()  
{  
    int k = 10;  
    cout << "k = " << k << "; k+3 = " << plus_3(k) << endl;  
    return 0;  
}
```

A saída do programa é como mostrado abaixo.

```
k = 10; k+3 = 13
```

Exercício:

Acrescente à biblioteca plus_n, feita na prática anterior, as funções plus_6 e plus_7. Altere o programa principal para que use uma dessas duas funções para testar a nova biblioteca.

6.6 Regras do compilador

O compilador segue um conjunto pequeno de regras simples para fazer o seu trabalho de construir o programa executável. Nessa seção, fala-se desse pequeno conjunto de regras. O nome “compilador” é ligeiramente inadequado.

O nome mais correto do programa compilador seria programa “construtor” (*builder*), pois é isso (construir) que o programa compilador realmente faz. A construção é o procedimento que inclui “compilar” e “ligar (*link*)”.

- Quando analisa um código, o compilador faz uma das duas coisas:
 - Gera código.
 - Aprende uma informação, que poderá ser útil para gerar código mais tarde.
- O compilador somente pode gerar código a partir de identificadores que tenham sido previamente declarados.
- Ao definir-se uma função, também se está declarando-a. Mas ao declarar-se uma função, não se está definindo. Declarar uma função é o mesmo que escrever o seu protótipo.
- O compilador precisa conseguir compilar cada arquivo fonte (*.c ou *.cpp) em uma única passagem, de cima para baixo. Do contrário, ocorre erro de compilação.
- A construção de um executável é feita em duas etapas – compilação e ligação.
- Um projeto é a construção de um executável a partir de um conjunto de programas fonte (*.cpp). Para construir um projeto, cada fonte é compilado como se os demais não existissem. Em seguida, a etapa de ligação faz construir o executável final.
- Ao iniciar a compilação de cada arquivo fonte *.c ou *.cpp, o compilador somente reconhece as palavras reservadas e a gramática da linguagem.
- Os arquivos de cabeçalho (*header*) *.h ou *.hpp *não devem* fazer parte diretamente do projeto¹⁵, e portanto *não* geram arquivo objeto. Mas isso não quer dizer que esses arquivos não sejam importantes. A finalidade desses arquivos é serem incluídos em arquivos fonte *.c ou *.cpp.

¹⁵ O leitor deve ter percebido que usou-se o termo “não devem” e não “não podem”. Isso porque o compilador segue regras simples e geralmente não realmente verifica a extensão dos arquivos fonte. Contudo, é considerado má programação fazer constar um arquivo *.h diretamente no projeto. É também má programação incluir (usando #include) um arquivo *.c ou *.cpp dentro de um outro arquivo fonte de mesma extensão, como se fosse um arquivo *.h.

7 Linguagem C/C++

7.1 Chamada de função por referência e por valor

Numa função, as variáveis que são parâmetros e as variáveis internas da função são chamadas de variáveis automáticas, pois são automaticamente criadas no momento de chamada da função e automaticamente destruídas quando a função termina.

Há basicamente 2 formas de se qualificar as variáveis automáticas que são parâmetros de uma função.

- 1) Parâmetros passados por valor.
- 2) Parâmetros passados por referência.

Quando um parâmetro é passado por valor, a variável parâmetro dentro da função é uma **cópia**¹⁶ da variável que chamou a função. Portanto qualquer alteração que ocorra na variável valerá apenas dentro da função e com o fim desta, o programa que chamou a função não perceberá nada.

Quando um parâmetro é passado por referência, a variável que se passa como parâmetro é na realidade o ponteiro para a variável que chamou a função. O que interessa na prática é que as alterações que se faz na variável dentro da função são percebidas fora da função.

No exemplo abaixo, 3 funções são definidas. A função f1 tem passagem de parâmetros por valor. A função f2 tem passagem de parâmetros por valor, mas como está recebendo o ponteiro para uma variável (um objeto), significa que colocar dados na referência do ponteiro corresponde a colocar dados na variável original. Ou seja, é como se a passagem de parâmetros fosse por referência. Mas como o referenciamento dos ponteiros deve ser feito explicitamente (tanto dentro da função f2, como na linha “*x = 2.2;”, quando na chamada da função f2, como na linha “f2(&a);”), essa forma de se passar parâmetros pode ser considerada como “pseudo por referência”. A função f3 recebe o parâmetro como um lvalue do tipo em questão (no caso float). Portanto, trata-se de uma passagem por referência verdadeira, pois tanto dentro da função f3 quanto na chamada da função f3, o uso de lvalue é implícito. Para modificar f3 para que a passagem de parâmetros seja por valor, basta apagar uma letra, que é o “&” no protótipo de f3.

¹⁶ Cópia no sentido que, no caso de programação orientada a objeto, o construtor de cópia é chamado.

```

#include <iostream.h>
void f1(float z) // por valor
{
    z = 1.1;
}
void f2(float *x) // pseudo por referência
{
    *x = 2.2;
}
void f3(float &v) // verdadeiro por referência
{
    v = 3.3;
}
int main()
{
    float a = 10.1;
    cout << "a=" << a << endl;
    f1(a);
    cout << "a=" << a << endl;
    f2(&a);
    cout << "a=" << a << endl;
    f3(a);
    cout << "a=" << a << endl;
    return 0;
}

```

Resultado:

```

a=10.1
a=10.1
a=2.2
a=3.3

```

Em C, a única forma de se passar parâmetros por referência é como mostrado em f2. Em C++, pode-se usar como mostrado em f3, que é mais elegante.

Os motivos para se optar por passar um parâmetro por valor ou por referência são vários (não apenas o fato de a função alterar ou não alterar o valor do parâmetro). Um deles trata do tempo de execução. Como as variáveis passadas por valor recebem uma **cópia** do seu valor quando é chamada, se essa variável é grande (pode ocorrer no caso de variáveis estruturadas, veja a seção 9.4.1) o tempo de cópia pode ser significativo, principalmente se essa chamada é feita num trecho repetitivo de programa, como um loop. Uma variável passada por referência passa somente o ponteiro da variável para a função, e o ponteiro tem valor fixo independente do tamanho da variável.

7.2 Tipos de dados definidos pelo programador

Além dos tipos padrão, o programador pode definir tipos próprios com diversas finalidades, que após definidos passarão a ser tão válidos como os demais. Pode-se, por exemplo, criar o tipo REAL que será usado no lugar dos tipos de ponto flutuante com finalidade de facilitar a substituição do tipo básico de ponto flutuante em todo o programa. No exemplo abaixo, foram definidas duas funções com parâmetros em ponto flutuante onde foi usado o tipo do usuário definido como REAL. que corresponde ao tipo padrão float. Caso se deseje substituir em todo o programa, para efeito das funções definidas pelo usuário, o

tipo float pelo tipo double, basta trocar a definição do tipo do usuário REAL de float para double. Veja o exemplo abaixo.

```
typedef float REAL;    // defino o tipo real
REAL myinverse(REAL x) // declaro e defino uma função usando o meu tipo REAL
{
    return (1./x);
}
REAL mydoubleinverse(REAL x) // outra função com o tipo REAL
{
    return (1./x/x);
}
```

Também é considerado tipo do usuário (*data type*) uma lista de identificadores com a palavra reservada `enum`. Por exemplo:

```
typedef enum days {sun,mon,tue,wed,thu,fri,sat}; // cria o tipo days
days today; // cria ocorrência do tipo days com today

ou em português
typedef enum dias {dom,seg,ter,qua,qui,sex,sab}; // cria o tipo dias
dias hoje; // cria ocorrência do tipo dias com hoje
```

7.3 Maquiagem de tipos (*type casting*)

Muitas vezes é preciso converter um tipo para outro. Por exemplo: `int` para `float`, etc. Em alguns casos o compilador sabe que tem que fazer a conversão e faz automaticamente para você. Algumas vezes a conversão não é feita automaticamente mas pode-se forçar a conversão maquiando o tipo. Para fazer a maquiagem, basta colocar antes da variável ou função em questão o novo tipo entre parênteses. No exemplo abaixo, a maquiagem de tipo garante a divisão em ponto flutuante.

```
#include <iostream.h>
int main()
{
    float f;
    f = 2/5; // divisão de ints
    cout << "f = " << f << endl; // resultado errado
    f = (float)2/(float)5; // type cast garante divisão de floats
    cout << "f = " << f << endl; // resultado certo
    return 0;
}
```

Muitos autores estão recomendando evitar o uso direto de maquiagem de tipos, como mostrado no programa acima. É mais seguro a utilização de `static_cast` (palavra reservada), como mostrado no programa abaixo. Contudo, apenas os compiladores mais modernos dão suporte a `static_cast`.

```
#include <iostream.h>
int main()
{
    float f;
    f = 2/5; // divisão de ints
    cout << "f = " << f << endl; // resultado errado
    // type cast garante divisão de floats
    f = static_cast<float>(2)/static_cast<float>(5);
    cout << "f = " << f << endl; // resultado certo
    return 0;
}
```

```
}
```

Em ambos os programas, a saída é como mostrado abaixo.

```
f = 0
f = 0.4
```

7.4 Operações matemáticas

Para se escrever operações matemáticas básicas, basta escreve-las no programa da mesma forma como no papel, respeitando-se a precedência dos operadores da mesma forma também.

```
double z,y,x;
y=3.33; z=4.5677;
x=y*y+z*3-5;
```

A biblioteca padrão oferece também inúmeras funções matemáticas de interesse geral, listadas abaixo. Para maiores informações sobre essas funções, consulte qualquer referência ou o help do compilador.

Funções matemáticas da biblioteca padrão de C

acos	asin	Atan	atan2	sin	tan
cosh	sinh	Tanh	exp	frexp	ldexp
log	log10	Modf	pow	sqrt	ceil
fabs	floor	Fmod			

7.5 Controle de fluxo do programa

Em todos os casos o teste do controle de fluxo, que é interpretado como booleano, é qualquer variável ou função de qualquer tipo (inclusive ponteiro). 0 significa falso e 1 significa verdadeiro.

Em todos os lugares onde há `<statement>` pode-se substituir por `<block statement>`, que é um conjunto de *statements* dentro de um escopo de chaves.

```
{ // block statement
    <statement1>;
    <statement2>;
}
```

Exemplo:

```
void fun2() { /* ... */ }
void fun3() { /* ... */ }
void fun4() { /* ... */ }

// uma maneira de se chamar 3 procedimentos em seqüência
void fun1()
{
    if (i == 1)
    {
        fun2();
        fun3();
    }
}
```

```

        fun4();
    }
}

// função auxiliar
void fun_auxiliary()
{
    fun2();
    fun3();
    fun4();
}

// outra maneira de se chamar 3 procedimentos em sequência
void fun1_new()
{
    if (i == 1)
        fun_auxiliary();
}

```

7.6 Execução condicional

```

if (<variavble>) <statement>
else <statement>;    // optional

```

Exemplo:

```

#include <iostream.h>
float a, b, c; // variáveis globais
int main()
{
    a = 2;
    b = 1.5;
    if (b == 0)
        cout << "c is undefined" << endl;
    else
    {
        c = a / b;
        cout << "c = " << c << endl;
    }
    return 0;
} // fim da função main

```

7.7 Laços (loop) de programação

Os laços de programação são estruturas de uma linguagem de programação para comandar instruções que se repetem num programa. Há 3 estruturas de laço em C++, mostradas abaixo.

7.7.1 Laço tipo “do-while”

```

do
    <statement>;
while (<variable>);

```

Exemplo:

```

#include <iostream.h>
int main()
{
    int i = 5;

```

```

do
{
    cout << "i = " << i << endl;
    i--; // i = i - 1
}
while (i>0);
return 0;
}

```

Resultado:

```

i=5
i=4
i=3
i=2
i=1

```

7.7.2 while

```

while (<variavble>)
    <statement>;

```

Exemplo:

```

#include <iostream.h>
int main()
{
    int i = 5;
    while (i>0)
    {
        cout << "i = " << i << endl;
        i--; // i = i - 1
    }
    return 0;
}

```

Resultado:

```

i=5
i=4
i=3
i=2
i=1

```

7.7.3 for

```

for (<inic> ; <test> ; <set>)
    <statement>;

```

```

/*===== o mesmo que
{
    <inic>;
    while (<test>)
    {
        <statement>;
        <set>;
    }
}
=====*/

```

Exemplo:

```

#include <iostream.h>
int main()
{

```



```

    for (int i = 5; i>0; i--)
        cout << "i = " << i << endl;
    return 0;
}

```

Resultado:

```

i=5
i=4
i=3
i=2
i=1

```

Há uma sutil porém importante incompatibilidade entre compiladores referente a palavra reservada “for”. Veja a seção 20.1.1 na página 429.

7.7.4 Alterando o controle dos laços com break e continue

Caso se inclua a palavra reservada “break” (parar) num laço controlado, o fluxo de execução é quebrado e o programa sai do laço que está executando. No exemplo abaixo, há dois laços do tipo for, e o break é ativado numa condição específica (j==3). Quando isso ocorre, o laço interno é quebrado, mas a execução continua no laço externo.

```

#include <iostream.h>
int main()
{
    for (int i = 0; i < 2; i++)
    {
        for (int j = 5; j>0; j--)
        {
            if (j == 3)
                break;
            cout << "i = " << i << ", j = " << j << endl;
        }
    }
    return 0;
}

```

Resultado:

```

i = 0, j = 5
i = 0, j = 4
i = 1, j = 5
i = 1, j = 4

```

A palavra reservada “continue” tem o significado de mandar o laço de programação corrente se encerrar (mas não se quebrar). Isto é, mesmo que haja mais comandos a serem executados antes do fim do laço, o laço se encerra caso encontre a palavra reservada “continue”. Se o exemplo acima substituir a palavra reservada “break” por “continue”, o que ocorre é que caso j==3, o laço interno se encerra, isto é, passa para j=4. Com isso, a linha que exterioriza o valor de j=3 não é executada. Portanto a saída é como mostrado abaixo.

```

i = 0, j = 5
i = 0, j = 4
i = 0, j = 2
i = 0, j = 1
i = 1, j = 5
i = 1, j = 4

```

```
i = 1, j = 2
i = 1, j = 1
```

7.7.5 Exercício

Prática: Fazer um programa para escrever a tabuada de 5.

Solução:

```
#include <iostream.h>
int main()
{
    int tab = 5;
    for (int i = 1; i <= 10; i++)
    {
        cout << i << "*" << tab << "=" << i*tab << endl;
    }
    return 0;
}
```

Prática:

Fazer um programa para escrever as tabuada de 3 a 6.

Solução:

```
#include <iostream.h>
void imprime_tabuada(int tab)
{
    cout << "-----" << endl;
    for (int i = 1; i <= 10; i++)
    {
        cout << i << "*" << tab << "=" << i*tab << endl;
    }
}
int main()
{
    int tab_min = 3;
    int tab_max = 6;
    for (int i = tab_min; i <= tab_max; i++)
        imprime_tabuada(i);
    return 0;
}
```

O Visual C++ possui uma sutil diferença na implementação do for, como mostrado na seção 20.1.1 (página 429).

7.8 switch-case

O switch, case e default (implícito) são palavras reservadas que servem para definir múltiplo redirecionamento do fluxo do programa. Frequentemente também se usa a palavra reservada `break`. Heis a sintaxe:

```
switch (<expr>)
{
    case <item>:
        <statements>;
    case <item>:
        <statements>;
    default:
        <statements>;    // quando nenhum dos itens coincide
}
```

Exemplo:

```
#include <iostream.h>
enum days {sun, mon, tue, wed, thu, fri, sat}; // cria o tipo days
days today; // cria ocorrência do tipo days com today
void m()
{
    today = sat;
    switch (today)
    {
        case mon:
        case tue:
        case wed:
        case thu:
        case fri:
            cout << "Vá para o trabalho" << endl;
            break;
        case sat:
            cout << "Limpe o jardim e ";
        case sun:
            cout << "relaxe." << endl;
            break;
        default:
            cout << "Erro, dia não é definido";
            // quando nenhum dos itens coincide
    }
}
int main()
{
    m();
    return 0;
}
```

7.9 arrays

Os arrays estáticos são declarados como:

```
<tipo_do_array> <nome_do_array>[<dimensão>];
```

Um array reserva espaço na memória para <dimensão> elementos do tipo <tipo_do_array> desde a posição [0] até a posição [<dimensão> - 1].

Exemplo:

```
#include <iostream.h>
int main()
{
    float f[5];
    for (int i = 0; i < 5; i++)
    {
        f[i] = 1 + i / 10.0;
        cout << f[i] << endl;
    }
    return 0;
}
```

Resultado:

```
1
1.1
1.2
```

1.3
1.4

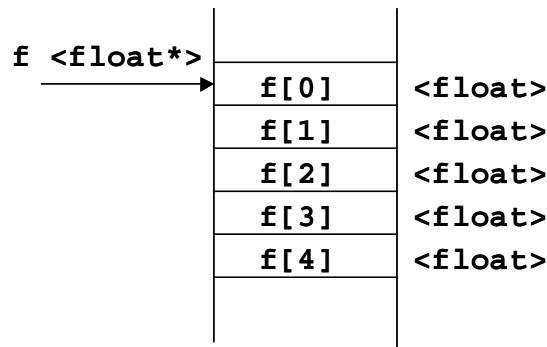


Figura 11: Visualização gráfica do array na memória

No exemplo acima, o identificador “f” é uma constante do tipo float*, ou seja, ponteiro para float. Ao declarar float f[5], o compilador reserva espaço para 5 variáveis do tipo escolhido float e o próprio identificador “f” é o ponteiro constante float*, que aponta para o início das 5 variáveis.

Ao usar uma das variáveis do array com o uso do operador [], o que realmente se está fazendo é uma referência do ponteiro somado ao índice escolhido, conforme será explicado na seção abaixo.

Ao se utilizar um array, não há *range-check*, isto é, checagem de tentativa de acesso fora da dimensão máxima do array.

```
#include <iostream.h>
int main()
{
    double valores[4]; // define-se um array
    // carrega-se valores no array
    valores[0] = 1.0;
    valores[1] = 1.1;
    valores[2] = 1.2;
    valores[3] = 1.3;

    for (int i = 0; i < 4; i++)
        cout << "valores[" << i << "] = " << valores[i] << endl;
    return 0;
}
```

Resultado:

```
valores[0] = 1.0;
valores[1] = 1.1;
valores[2] = 1.2;
valores[3] = 1.3;
```

No exemplo abaixo, define-se um array de strings com 4 posições, devidamente carregado com constantes. Nesse caso, existem as posições [0], [1], [2] e [3]. Em seguida, executa-se um laço que manda imprimir o conteúdo do array, mas o programa comanda o laço indo até a posição [4]. Isso não gera um erro de

compilação ou de execução, mas é um erro de lógica, pois o valor do array na posição [4] não é definido.

Exemplo:

```
#include <iostream.h>
int main()
{
    const char* nomes[4] = {"Manuel", "Joaquim", "Maria", "Bruno"};
    for (int i = 0; i < 5; i++)
        cout << "Nome[" << i << "] = " << nomes[i] << endl;
    return 0;
}
```

Resultado:

```
Manuel
Joaquim
Maria
Bruno
sldflaskf j;alsdkj (indefinido)
```

7.10 Ponteiros

O uso de ponteiros são um recurso muito poderoso da linguagem C/C++. O fato de ser um recurso poderoso é ao mesmo tempo uma vantagem e uma desvantagem. O grande poder faz aumentar a probabilidade de bugs e defeitos nos programas. Além disso, há um certo clima de mistério em torno do uso de ponteiros.

Creio que a melhor forma de se compreender o uso de ponteiros é começar pelo começo, isto é, relembrar como a CPU se une eletronicamente com a memória ou outro periférico. A figura abaixo ilustra essa ligação, de forma praticamente igual para quase todas as CPUs existentes. Um parâmetro importante num computador é o número de fios do barramento de endereços da CPU. Esse número de fios é o tamanho do ponteiro para a CPU em questão. Num computador tipo PC, esse número é 32. Num chip microcontrolador, os números típicos são 8, 16 ou 32. Um computador de porte maior que um PC pode ter barramento de 64 ou 128 bits, ou até mais.

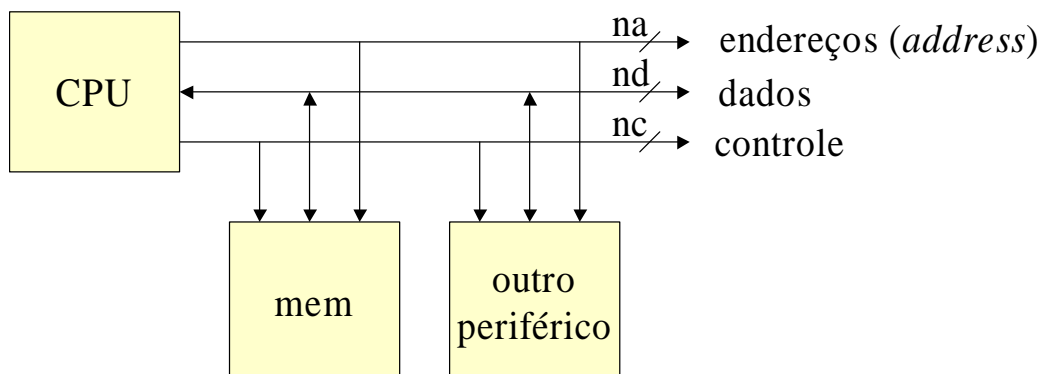


Figura 12: Diagrama ilustrando como a memória ou outro periférico liga-se a CPU

O tamanho de um ponteiro é relacionado apenas ao número de fios no barramento de endereços da CPU. Não há relação entre o tamanho do ponteiro e o tamanho do objeto ou variável que ele aponta. O ponteiro pode apontar para dados ou para código. Apontar para dados é apontar para variáveis (objetos). Apontar para código é apontar para uma função.

O programa abaixo é um exemplo simples do uso de ponteiros. A função main tem 2 variáveis locais. Uma tem o identificador “f”, e é do tipo float. A outra tem o identificador “pf” e é do tipo float*, isto é, ponteiro para float. Inicialmente carrega-se a variável “f” com a constante 1.1. O operador “&” faz retornar o endereço da variável em questão. Portanto, a linha “pf = &f;” pega o endereço da variável “f” e a armazena em “pf”, que tem justamente o tipo adequado para armazenar esse tipo de informação. Em seguida, carrega-se a constante 2.2 na referência do ponteiro (“*pf” significa “no lugar que pf está apontando, ou seja, a referência do ponteiro”). Como pf foi carregado com o endereço da variável f, a constante 2.2 será na realidade armazenada em f. Em seguida, mostra-se no console o conteúdo da variável f com o valor 2.2. O valor de pf, um número de 32 bits também é mostrado.

```
#include <iostream.h>
int main()
{
    float f = 1.1;
    float *pf;
    pf = &f;
    *pf = 2.2;
    cout << "f=" << f << endl;
    cout << "pf=" << pf << endl;

    return 0;
}
```

Resultado:

```
f=2.2
pf=0x0066FDF4
```

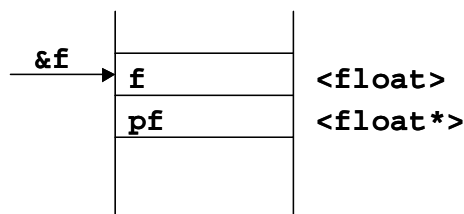


Figura 13: Diagrama ilustrando a memória com o float e o ponteiro para float

7.10.1 Ponteiro para ponteiro

Um variável do tipo ponteiro é uma variável como outra qualquer. Portanto, um outro ponteiro pode apontar para uma variável do tipo ponteiro. Nesse caso

trata-se de “ponteiro para ponteiro”. Não há limite para o número de vezes que um ponteiro aponta para um ponteiro.

No exemplo abaixo, a função main possui 4 variáveis locais, cujos identificadores são f, pf, ppf, pppf. A primeira é um float. A segunda um ponteiro para float. A terceira um ponteiro para ponteiro para float, e a quarta um ponteiro para ponteiro para ponteiro para float. O programa armazena adequadamente o endereço de um ponteiro no ponteiro para ponteiro correspondente. A linha “*pf=2.2” deve ser entendida como “coloque 2.2 na variável tipo float cujo endereço indicado por pf”. A linha “**ppf=3.3” deve ser entendida como “coloque 3.3 na variável tipo float cujo endereço indicado pelo ponteiro indicado por ppf”. Analogamente, a linha “***pppf=4.4” deve ser entendida como “coloque 4.4 na variável tipo float cujo endereço indicado pelo ponteiro indicado pelo ponteiro indicado por pppf”. Apesar de tantos ponteiros, somente há um lugar onde se pode armazenar um float no programa abaixo.

```
#include < iostream.h>
int main()
{
    float f = 1.1;
    float *pf = &f;
    float **ppf = &pf;
    float ***pppf = &ppf;
    *pf = 2.2;
    cout << f << endl;
    **ppf = 3.3;
    cout << f << endl;
    ***pppf = 4.4;
    cout << f << endl;

    return 0;
}
```

Resultado:

```
2.2
3.3
4.4
```

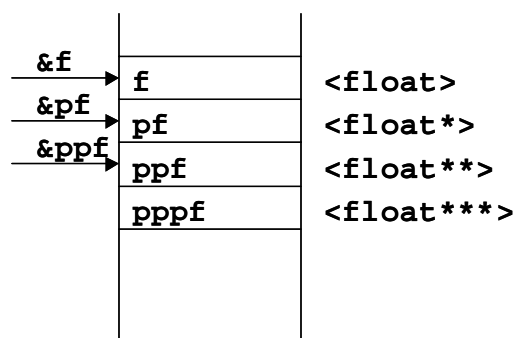


Figura 14: Diagrama ilustrando a memória com o float e os ponteiro para ponteiros para float

Atenção na sintaxe para ponteiros. Na linha de programa abaixo, “d” é um double, “pd” é um ponteiro para double e “ppd” é um ponteiro para ponteiro para double.

```
double d, *pd, **ppd;
```

Um erro comum é esquecer de colocar o asterisco. Na linha abaixo, “a” é um ponteiro para double, “b” é um double, e “c” é um outro double.

```
double *a, b, c;
```

7.11 Arrays e ponteiros

7.12 Arrays multidimensionais

São muito semelhantes aos arrays de 1 dimensão. Para a definição, apenas acrescenta-se a segunda dimensão após a primeira. A utilização é feita da mesma forma que a definição, com múltiplas indireções (por exemplo [i][j]).

Na declaração de um array estático, caso se queira carrega-lo com um valor de inicialização na compilação, pode-se declarar o array omitindo-se a primeira dimensão, pois os dados declarados a seguir contém esta informação. Todas as dimensões exceto a primeira precisam ser declaradas. Veja o exemplo:

Exemplo:

```
void fun()
{
    // aqui também se exemplifica como inicializá-lo na compilação
    int a[][4] = { {1, 2, 3, 4} , {5, 6, 7, 8} , {9, 10, 11, 12} };
    int b[][4][5] = { {1, 2, 3, 4} , {5, 6, 7, 8} , {9, 10, 11, 12} };

    int i = a[1][1]; // uso de array multivariável
}
```

7.13 Parâmetros da função main

Na verdade, o programa gerado por um compilador C / C++ deve ser entendido como o código fonte abaixo:

```
// inicializações do compilador
exit_code = main (argc, argv);
// finalizações do compilador
```

A função `main` é aquela que chama o próprio programa, conforme já foi explicado. Esta função admite ter parâmetros (argumentos) que são as palavras escritas na linha de comando que chamou o programa. O protótipo da função `main` é:

```
int main (int argc, char *argv[]);
```

O retorno da função `main` é o código de retorno ao sistema operacional, que só interessa para que um arquivo *.bat do DOS (em programas para Windows), ou um script em um programa em unix, mas isso não será abordado aqui.

O primeiro argumento `int argc` é o número de parâmetros passados ao programa (incluindo o próprio nome dele) e o segundo `char **argv` é um array de strings (`char*`) cujos conteúdos são exatamente os parâmetros passados. Veja o exemplo.

Exemplo:

```
#include <iostream.h>
int main(int argc, char *argv[])
{
    for (int i = 0; i < argc; i++)
        cout << "o argumento de chamada numero " << i << " é " << argv[i] << endl;

    return 0;
}
```

Compile o código acima, chame-o de `prog`, por exemplo. Verifique o seu funcionamento utilizando a linha de comando abaixo:

```
prog a1 a2 a3 a4 a5 "one sentence"
```

O resultado será:

```
o argumento de chamada numero 0 é C:\USERS_1\VILLAS_C\BORLAND_C\T3.EXE
o argumento de chamada numero 1 é a1
o argumento de chamada numero 2 é a2
o argumento de chamada numero 3 é a3
o argumento de chamada numero 4 é a4
o argumento de chamada numero 5 é a5
o argumento de chamada numero 5 é one sentence
```

É muito comum que um programa seja usado a partir dos parâmetros passados pela linha de comando e nestes casos é de bom alvitre dar informações rápidas sobre a utilização do programa quando não se passa parâmetro algum. Veja o exemplo de uma estrutura genérica de um programa deste tipo.

Exemplo:

```
#include <iostream.h>
int main(int argc, char *argv[])
{
    // prints the always seen message
    cout << "Prog" << endl;
    cout << "Version 1.3 - by Author, in Jan 20th, 1998" << endl;

    if (argc < 2)
    {
        // called with no arguments. Display immediate help
        cout << "Usage: Prog [-options] def[.def]"
              << " source[.ex1] [destin[.ex2]]";
        cout << "Something Else ..." << endl;
    }
    else
    {
        // do what the program should do
        // do_the_thing(argv[1],argv[2]);
    }
    return 0;
}
```

7.14 Compilação condicional

A diretiva `#if` e `#endif` compila condicionalmente um trecho de programa. A sintaxe é mostrada abaixo.

```
#if <condição>
// trecho de programa
#endif

#if 0
    int pi=3.1416;
#endif
```

7.15 Pré processador e tokens (símbolos) usados pelo pré-processador

A linguagem C / C++ compila os arquivos fonte após esses arquivos serem pré-processados a partir de algumas diretivas, iniciadas pelos símbolos abaixo.

##

```
#define PI 3.1416
#ifndef PI
int pi=3.1416;
#endif
```

As diretivas de pré-compilação mais importantes são:

7.16 #define

O `#define` é uma diretiva que troca seqüências de texto no código fonte, para em seguida esse texto ser compilado. O uso dessa diretiva também é chamada de macro. É um estilo recomendável de boa programação que as macros sejam escritas com maiúsculas.

```
#define PI 3.1416
float area_circulo(float raio) {
    return raio*raio*PI; // expandido para return raio*raio*3.1416;
}
```

O `#define` pode ser parametrizado. Veja o exemplo abaixo.

```
#define FORMULA(x,y) (x)*3 + (y)*4 + 5
int main () {
    float r = FORMULA(1.2,3.4); // expandido para r = (1.2)*3 + (3.4)*4 + 5;
    return 0;
}
```

É recomendável que se escreva a parametrização das macros usando parêntesis envolvendo os parâmetros, para se evitar erros como no exemplo abaixo.

```
#define FORMULA(x) x*3
int main()
{
    float r = FORMULA(1 + 2); // expandido para r = 1+2*3,
```

```

    // que é diferente de (1+2)*3
    // melhor seria escrever a macro como #define FORMULA(x) (x)*3
    return 0;
}

```

7.17 operador

Para a construção de strings a partir de parâmetros dentro de um #define, usa-se o operador #

```

#define PATH(logid,cmd) "/usr/" #logid "/bin/" #cmd

char *mytool = PATH(francisco,readmail);
// expandido para: char *mytool = "/usr/" "francisco" "/bin/" "readmail";
// é equivalente a: char *mytool = "/usr/francisco/bin/readmail";

```

7.18 operador

O operador ## é uma diretiva para o pré processador da linguagem C/C++. Esse operador aparece como um token de troca entre 2 tokens, concatenando-os. Veja o exemplo abaixo.

```

#include <iostream.h>
#define TIPO(a) FUT ## a
#define FUTSAL "futebol de salão"
#define FUTBOL "tradicional futebol de campo"

int main ()
{
    cout << TIPO(SAL) << endl;
    cout << TIPO(BOL) << endl;
    return 0;
}

```

resulta em:

```

futebol de salão
tradicional futebol de campo

```

7.19 Número variável de parâmetros

Uma função pode ter número variável de parâmetros. Mas para usar esse recurso, a função deve de alguma forma saber quantos parâmetros serão chamados. Veja o exemplo abaixo.

```

#include <iostream.h>
#include <stdarg.h>

double sum_parameters(int num,...)
{
    double sum = 0.0;
    double t;
    va_list argptr;
    va_start(argptr,num); // initialize argptr
    // sum the parameters
    for ( ; num ; num--) {
        t = va_arg(argptr,double);
        sum += t;
    }
    va_end(argptr);
    return sum;
}

```

```

}

int main ()
{
    cout << "the sum is " << sum_parameters(2, 1.2, 3.4) << endl;
    cout << "the sum is " << sum_parameters(3, 1.2, 3.4, 4.4) << endl;
    return 0;
}

```

A saída do programa é como mostrado abaixo.

```

the sum is 4.6
the sum is 9.3

```

7.20 Exercícios

- 1) Avalie os arquivos fonte abaixo (*.cpp). Para cada um deles, diga se o arquivo compila ou não. Caso o arquivo não compile, diga qual o erro que o compilador gera. Para cada arquivo que não compila, faça uma adaptação (simples) para que compile.
- 2) Todos os arquivos fonte (*.cpp) abaixo compilam. Considere cada um dos projetos abaixo, e diga quais geram executáveis, e quais geram erros de link.

C++ Multiplataforma e Orientação a Objetos

Parte 3: C++ e Orientação a Objetos

8 Técnicas para melhoria de rendimento em programação

8.1 Reutilização de código

O trabalho de programação é um trabalho caro. Projetos grandes na área de software requerem uma quantidade muito significativa de homem-hora e com isso a produção de um programa pode representar um investimento vultoso. Além disso, a crescente complexidade do software torna praticamente impossível que apenas um indivíduo possa desenvolvê-lo completamente. Até mesmo software de “média complexidade” tende a ser feito por um time.

Uma das chaves para se melhorar o rendimento da programação é o reaproveitamento de “blocos de código”, também conhecido como “componentes”. Concretamente isso significa ser capaz de usar uma biblioteca feita por outra pessoa ou de entregar para outra pessoa uma biblioteca desenvolvida por você. A biblioteca, no caso, é um conjunto de componentes prontos para serem reaproveitados.

Para que se possa aproveitar trabalho de outros e permitir que outros usem nosso trabalho, é preciso entre outras coisas definir uma nomenclatura para o que se está trabalhando. Para isso, define-se os termos abaixo.

1. **Componente** - Um componente de software é um bloco de código que pode ser usado de diversas formas dentro de um programa. Por exemplo: listas, arrays e strings são componentes que podem ser usados em muitos programas. Também são considerados componentes de software elementos de interface, como check box, ou radio button.

Os componentes devem ser escritos como uma função de propósito genérico, de forma que possa ser usado das mais variadas formas. Quem desenvolve uma aplicação e quer usar um componente de software, não precisa saber como é o funcionamento interno desse componente.

Tipicamente “descobre-se” um componente quando se está programando e percebe-se que há trechos de programação que estão se repetindo naturalmente. É como se algo estivesse pedindo que todos esses trechos se transformassem num componente genérico que possa ser usado sempre que aquela necessidade surja.

2. **Biblioteca (*library*)** - É um conjunto de componentes de software prontos para serem usados pelo programador.

3. **Estrutura (*framework*)** - Um estrutura é um esqueleto de programa contendo algum código e indicação para a introdução de código extra. Uma estrutura é usada para se gerar uma aplicação com determinadas características.

Uma estrutura deve ser de fácil entendimento para que um programador possa usa-la para fazer um aplicativo. Um exemplo de estrutura é um conceito usado pelo VisualC, chamado “application wizard”, que cria uma estrutura pronta para o programador.

4. **Aplicação** - uma aplicação é um programa completo. Os exemplos são inúmeros - editores de texto, jogos, planilhas, etc. Uma aplicação frequentemente baseia-se numa estrutura e usa diversos componentes.

Seja um projeto de software cuja especificação requer que se faça gráficos a partir de dados que vem de um sensor. Nesse caso, seria muito útil ter em mãos uma biblioteca pronta com componentes que fazem gráficos de forma genérica a partir de dados também genéricos. Assim, o programador precisaria apenas escrever a parte do código para ler os dados do sensor em seguida chamar os componentes da biblioteca gráfica para que os gráficos fossem feitos.

Conclusão: deve-se pensar em programação sempre considerando se um trecho de programa é de aplicação genérica ou de aplicação específica. Caso se perceba que uma funcionalidade do programa é de aplicação genérica, essa funcionalidade é candidata a um componente.

No exemplo acima, as funções de uso do sensor são (a princípio) específicas. São também específicas os detalhes de como os dados devem ser mostrados no gráfico. Mas as funções que desenham gráficos são genéricas, pois desenham qualquer coisa. Nesse sentido, faz sentido colocar as funções gráficas como componentes numa biblioteca, que poderia se chamar “grafico”. Qualquer novo problema que precise de gráficos poderia usar esses mesmos componentes.

Caso o sensor que se está usando seja ligado a uma placa de aplicação genérica acoplada ao computador, poderá ser uma boa idéia que se faça uma biblioteca também para as funções de uso desta placa. Assim, qualquer outro projeto que precise usar a tal placa com sensor poderá partir de funções de uso básico do sensor.

Não se esqueça: *programar é usar e desenvolver bibliotecas*. Portanto, antes de se iniciar um esforço de programação, pense se o problema em questão é genérico ou específico. Caso seja genérico, procure por uma biblioteca pronta, que possivelmente existe. Caso não exista uma, ou a que exista não seja satisfatória, considere desenvolver uma biblioteca para o problema genérico em questão. Assim, a próxima pessoa que pensar nesse assunto possa aproveitar parte do seu esforço de programação.

É importante também que se saiba encomendar um serviço de programação. Para um certo problema pode-se solicitar a terceiros (um profissional autônomo, uma empresa, etc) que se desenvolva uma biblioteca de forma a atender as necessidades do problema, contendo funções básicas. A partir de uma biblioteca, torna-se relativamente fácil fazer “em casa” (isto é, dentro da instituição em que se trabalha) a *customização* do programa que se está desenvolvendo. O termo *customização* é um inglesismo que vem de *customization*, isto é, fazer alguma coisa de acordo com o gosto do *custom* (cliente, freguês). Portanto, *customizar* um programa significa implementar ao detalhe as solicitações de quem solicita o trabalho. Por exemplo: escrever as telas de entrada e saída de dados, efeitos de multimedia, etc.

Pense neste exemplo: há uma placa com um sensor que se deseja usar. Pode-se solicitar a terceiros que desenvolva uma biblioteca, com fonte, contendo funções básicas de uso da placa com sensor — seleção do sensor, inicialização do sensor, leitura do sensor, etc. Tudo feito com tutorial ensinando a usar as funções desenvolvidas. A partir das funções da biblioteca, seria em princípio fácil se desenvolver um programa para usar a placa com sensor.

8.2 Desenvolvimento e utilização de componentes

Para que se possa desenvolver e utilizar componentes, é preciso se pensar se um problema a ser resolvido é específico ou genérico. É muito comum que um problema possa ser dividido em uma componente específica e uma componente genérica.

Por exemplo: Um programa precisa fazer uns gráficos a partir de dados que vem de um sensor. Os detalhes de como o gráfico deve ser (valor mínimo e máximo do eixo das abcissa e da ordenada, tipo do gráfico, etc) são um problema específico. Mas o traçado do gráfico em si é um problema genérico. Esse programa provavelmente pode ser resolvido com o seguinte ordenação:

1. Adquire-se os dados.
2. Chama-se a rotina gráfica (genérica) com parâmetros que tornem o gráfico exatamente como a especificação manda.

Como o mercado de software está muito desenvolvido, é bem possível que qualquer coisa que seja de interesse geral (como o traçado de um gráfico) já tenha sido desenvolvida e que haja várias bibliotecas disponíveis. Geralmente há bibliotecas algumas gratuitas e outras pagas. Inicialmente as gratuitas eram simples demais ou careciam de um manual ou help mais elaborado, e as pagas eram as que (supostamente) seriam completas e com suporte melhor.

Ultimamente, contudo, está se observando um fenômeno que alguns chamam de “software livre” ¹⁷.

Esse fenômeno se caracteriza pelo aparecimento de vários grupos que estão literalmente dando o seu trabalho de programação. O nível de qualidade do software desses grupos está crescendo muito, assim como o número de usuários. Em vários casos, o grupo dá o software e o texto fonte. Com isso, não há realmente nenhuma razão para não se experimentar usar o tal software e isso permite que o número de usuários cresça rápido. Os usuários logo se organizam em “tribos” para trocar informações, e isso corresponde a existência de um bom suporte para o produto.

Para o programador (ou para quem coordena programadores), é importante saber sobre o “software livre” para que se possa alocar otimamente o esforço de programação. O uso de uma biblioteca pronta, feita por um grupo externo ao seu, *pode* ser uma boa alternativa. Mas também pode não ser. A biblioteca é boa se ela resolve o problema em questão e se é sabido como usa-la (aliás, como qualquer software). É preciso que haja sentimento para decidir se é melhor tentar procurar por uma biblioteca pronta, aprender a usa-la, testa-la, etc, ou se é melhor desenvolver a biblioteca internamente, para que as especificações sejam atendidas perfeitamente. Essa é uma decisão difícil.

8.3 Programação estruturada

Para facilitar o desenvolvimento e uso de componentes reaproveitáveis de código, surgiu o conceito de programação estruturada. A programação estruturada consiste basicamente de se estimular o programador a usar “estruturas”, isto é, classes (class ou struct, veja sobre uso de classes na seção 9.4.1) que contém um conjunto de variáveis. Por exemplo: uma classe “pessoa” pode conter vários campos como nome, telefone, endereço, etc.

Com essa filosofia de trabalho, o programador iniciaria o seu trabalho definindo como devem ser as estruturas (tipos, classes) necessárias para se resolver um problema. Isto é, quais exatamente devem ser os campos que a classe “pessoa” deve ter, por exemplo. Em seguida, cuida de desenvolver um conjunto de funções para processar variáveis do tipo “pessoa”.

A programação estruturada já é um grande avanço para o aumento da produtividade da programação. Mas a experiência mostrou que alguns conceitos ainda faltavam. O desenvolvimento da tecnologia de software acabou produzindo um conjunto de conceitos ainda mais evoluídos que a programação estruturada. O uso desses novos conceitos ficou conhecido como programação “orientada a objeto”. Desde que esse texto vai abordar diretamente os conceitos

¹⁷ Em inglês é “free software”, que significa ao mesmo tempo “software livre” e “software gratuito”.

de orientação a objeto, creio que não convém “perder tempo” comentando sobre programação estruturada. É melhor ir direto para a programação orientada a objeto, que é mais fácil e mais poderosa.

9 Programação orientada a objeto

9.1 Conceitos básicos

Uma linguagem OO deve dar suporte a basicamente 3 conceitos – polimorfismo, encapsulamento e herança. Especificamente em C++, pode-se definir as características básicas de programação orientada a objetos, incluindo os sub-itens, como mostrado na lista abaixo. Nessa listagem, o conceito de “encapsulamento” aparece como um sub-item do conceito mais geral “objetos”.

- 1) polimorfismo
 - a. sobrecarga de operadores
 - b. argumento implícito (*default argument*)
- 2) objetos
 - a. uso de classes (funções dentro de estruturas)
 - b. construtor & destrutor
 - c. encapsulamento (ocultação de informação)
- 3) herança
 - a. ligação adiantada \times ligação tardia – (*early bind* \times *late bind*)
 - b. classe abstrata

Esses conceitos possuem significado isoladamente, e também podem atuar em conjunto com efeito em sinergia. A exploração desses conceitos é o objetivo desse capítulo, que será feito nas seções seguintes.

9.2 Nomenclatura para paradigma procedural e para paradigma OO

Uma das confusões na compreensão de programação orientada a objetos é o fato de a nomenclatura mudar em relação ao que geralmente se usa na programação procedural. Na tabela abaixo encontram-se as principais mudanças de nomenclatura

Programação procedural	Programação orientada a objetos
tipo, estrutura (struct)	classe
variável	objeto

(ocorrência de tipo)	(ocorrência de classe)
função	método (função membro, i.e. função dentro de uma classe)
dado	atributo (dado membro)

```

struct myStruct
{
    double d;
    int i;
};

void fun(myStruct z)
{
    // ...
}

void m()
{
    myStruct a;
    a.d = 1.1;
    a.i = 2;
    fun(a);
}

```

No programa acima, na nomenclatura procedural se diria que “mystruct” é um “tipo estruturado”. “a” é uma “variável” do tipo “mystruct”. “fun” é uma “função”, que processa os “dados” “a.d=1.1” e “a.i=2” armazenados no variável estruturada “a”.

Na nomenclatura orientada a objeto se diria que “mystruct” é uma classe, e “a” é um objeto (*instância* da classe “mystruct”). “d” e “i” são “atributos” da classe “mystruct”. Nesse caso, a classe não tem métodos (funções membro). “fun” é uma função global.

9.3 Representação gráfica de classes

Uma forma de se representar graficamente classes é utilizar um retângulo com bordas arredondadas, com 3 seções. Na parte de cima, escreve-se o nome da classe. Na seção seguinte, escreve-se os atributos da classe. Na última seção escreve-se os métodos da classe.

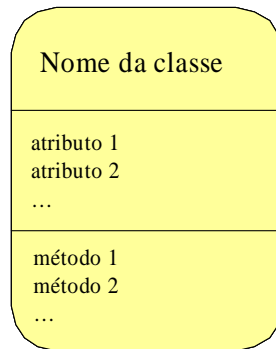


Figura 15: Representação gráfica de classe

Quando se deseja representar graficamente um objeto, pode-se usar o diagrama mostrado abaixo. O objeto é um retângulo de borda arredondada, com linha tracejada.

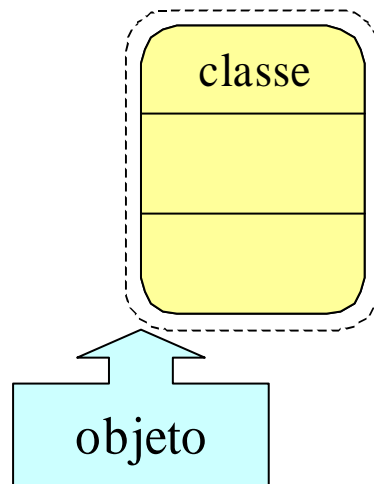


Figura 16: Um objeto com ocorrência de uma classe

9.4 Objetos

A programação estruturada enfoca a criação de estruturas (tipos estruturados) do usuário e de funções para manipular essas estruturas. Na programação orientada a objeto, as funções que manipulam os dados ficam dentro da mesma estrutura que contém os dados. Essa nova estrutura que contém dados e funções passou a ser chamada de objeto.

O fundamento da programação orientada a objetos é o uso de classes, que são descrições de objetos. A partir de uma classe pode-se *instanciar* inúmeros objetos. As classes são estruturas com dados e funções membros. “objeto”, para o programador, é uma “instanciação (ocorrência) de uma classe”. Já o analista de sistemas vê o “objeto” como uma abstração do mundo real, que interage com os demais objetos por sua interface. “aluno”, “produto”, “usuário”, são alguns nomes candidatos a serem classes.

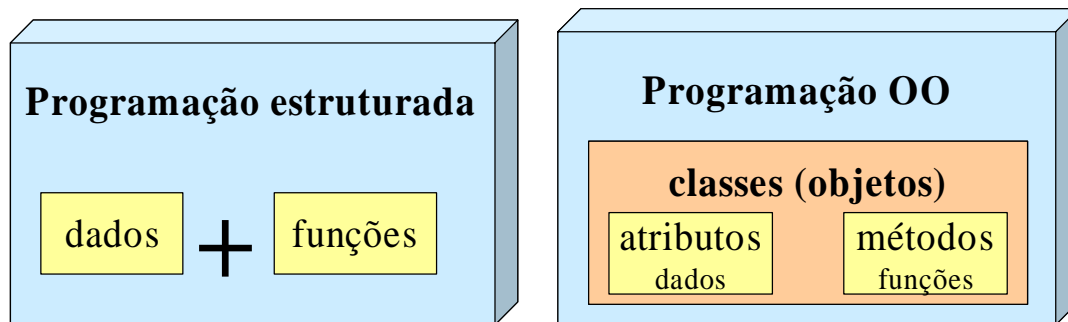


Figura 17: Visualização do paradigma estruturado e do paradigma OO

9.4.1 Uso de classes (funções dentro de estruturas)

Com a programação OO, uma estrutura pode ter funções dentro de estruturas. As estruturas tipicamente passam a ser chamadas de classes. As ocorrências (instâncias) das classes são os objetos.

```
class myClass
{ // início da definição de classe
public: // sem restrição de acesso
    int m_i; // dado membro (atributo)
    void set_i(int i)
    { // função dentro da classe (método)
        m_i = i;
    }
    int get_i()
    { // função dentro da classe (método)
        return m_i;
    }
}; // fim da definição de classe

int main()
{
    myClass z; // z é um objeto do tipo myClass
    z.m_i = 4; // acesso a dado membro público
    z.set_i(4); // chamando um método da classe myClass
    int k = z.get_i(); // chamando um método da classe myClass
    return 0;
}
```

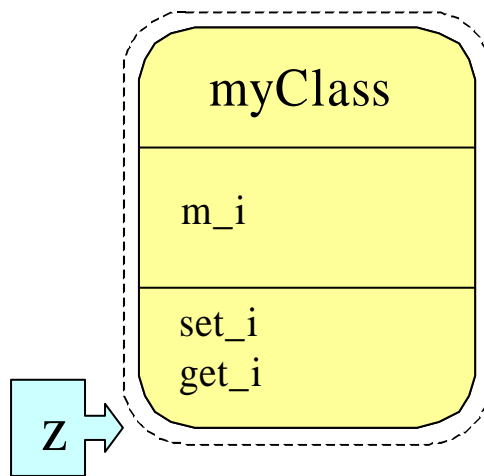


Figura 18: z é um objeto do tipo myClass

O objeto é, para todos os propósitos e finalidades, uma ocorrência (*instance*) de um tipo definido pelo usuário, em outras palavras: é uma variável. Pode parecer estranho para programadores tradicionais aceitar que um variável possa conter métodos (código executável) além de dados, mas é exatamente isso que ocorre na POO. Os objetos podem ser manipulados por funções em geral sendo passados como parâmetros ou sendo retornados.

Caso o objeto faça uso de alocação de memória, é necessário atenção especial no desenvolvimento da classe. Veja a seção 9.16.2.

9.4.2 Operador de escopo

Pode-se repetir o exemplo anterior, separando-se da classe o protótipo das funções da implementação das mesmas. Agora, há um projeto C++ em que existem dois arquivos fonte: main.cpp e myClass.cpp. Existe também um arquivo de cabeçalho myClass.h. No arquivo cabeçalho, escreve-se a classe em questão, sendo que os métodos estão escritos apenas a nível de protótipo. Apenas com essa informação, o arquivo main.cpp já pode compilar. Mas para que possa haver ligação de blocos e geração de código executável, é preciso que exista o arquivo myClass.cpp, em que se escreve a definição dos métodos da classe myClass. Note que um método de uma classe não é a mesma coisa que uma função global com mesmo protótipo. Para que se escreva a definição de um método fora da declaração da classe, é preciso que o protótipo do método se refira a classe em questão. Isso é feito com o operador de escopo (::), como mostrado no exemplo abaixo.

```

////////////////////////////////////
// myClass.h
class myClass
{
public:
    int m_i;
    void set_i(int i);
    int get_i();

```

```
};

////////////////////////////////////
// main.cpp
#include "myClass.h"
int main()
{
    myClass z;
    z.m_i = 4;
    z.set_i(4);
    int k = z.get_i();
    return 0;
}

////////////////////////////////////
// myClass.cpp
#include "myClass.h"
void myClass::set_i(int i)
{
    m_i = i;
}
int myClass::get_i()
{
    return m_i;
}
```

9.5 Polimorfismo

Polimorfismo significa, pelo radical da palavra, “muitas formas”. Em programação quer dizer que o mesmo nome pode ser dado a funções diferentes, desde que o contexto possa discernir de qual se trata pelo *prototype*. Por exemplo: uma função de inicialização de variáveis globais pode preencher com zeros (se não é passado parâmetro) ou com o valor do parâmetro passado. Veja !

Exemplo:

```
int x1,x2,x3; // variáveis globais a inicializar
void inicializa() { // chamada sem parâmetros
    x1=x2=x3=0; // preenche variáveis com zeros
}
void inicializa(int value) { // chamada com parâmetro
    x1=x2=x3=value; // preenche variáveis com parâmetro passado
}
```

Neste caso, a função “inicializa” pode ter dois códigos diferentes. O compilador determina em tempo de compilação qual é a que será usada pelo *prototype*. Por exemplo:

```
inicializa(); // chama a função de cima
inicializa(3); // chama a função de baixo
```

9.5.1 Argumento implícito (*default argument*)

Neste caso específico, as duas funções poderiam ser uma só, com a inicialização na linha de *prototype* do código da função. A inicialização de `value` só ocorre se

não se passar parâmetro. Isto se chama argumento implícito (*default argument*). Veja:

Exemplo:

```
void inicializa(int value=0) { // chamada sem parâmetro implica value=0
    x1=x2=x3=value;          // preenche variáveis com value
}
```

9.6 Análise, projeto e programação OO

“OO” é uma filosofia de trabalho, um paradigma. A linguagem de programação que se usa precisa dar suporte a OO para que se possa trabalhar nessa filosofia. C++ é uma das linguagens que dá suporte a OO. Optar ou não pelo paradigma OO é decisão que afeta o trabalho de desenvolvimento e manutenção de software, mas em princípio não faz diferença para o usuário final. Intuitivamente trabalha-se com paradigma procedural. O paradigma OO precisa ser estudado para ser compreendido.

Um projeto de software que adote o paradigma OO tem como característica a transposição das abstrações da análise quase diretamente para as classes da programação. Um software desenvolvido sem projeto formal tem também várias vantagens pela utilização de OO. Em C++, pode-se partir desde o baixo nível de aplicação e na própria linguagem definir-se classes que fazem o nível de programação crescer até um nível arbitrário. Com isso, pode-se desenvolver bibliotecas que tornam a programação bastante poderosa, com alto nível de reutilização de código. A biblioteca MFC, da Microsoft, é um exemplo de como se pode tornar alto o nível de programação GUI, que é intrinsecamente bastante complexa. A partir do uso de classes da biblioteca MFC, é relativamente fácil fazer um aplicativo Windows GUI com funcionalidade bastante sofisticada.

9.7 Exemplo conceitual sobre herança

Seja uma análise de um quadro funcional de uma empresa, onde se pode separar os integrantes em 4 classes - presidente, gerente financeiro, gerente comercial e secretária. As características deles estão no quadro central da figura abaixo e os métodos que eles exercem estão no quadro inferior. Por exemplo: o presidente tem por característica “salário e “tem_carro”, e seu método é “decide”.

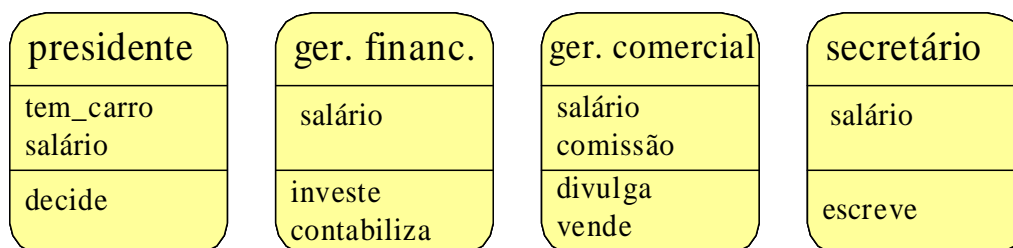


Figura 19: Visão gráfica das entidades (classes) de uma empresa

Como todos têm salário, pode-se conceber uma classe que possua esta característica, chamada “funcionário” por exemplo, que levará esta propriedade às demais por herança. Neste caso a figura ficará como no quadro abaixo, já assinalado com as ocorrências das classes. Os “objetos” são as ocorrências das classes. Nesse caso, o Paulo é o presidente, a Ana é a gerente financeira, o Fernando é o gerente comercial, e Sônia e Marcos são secretários.

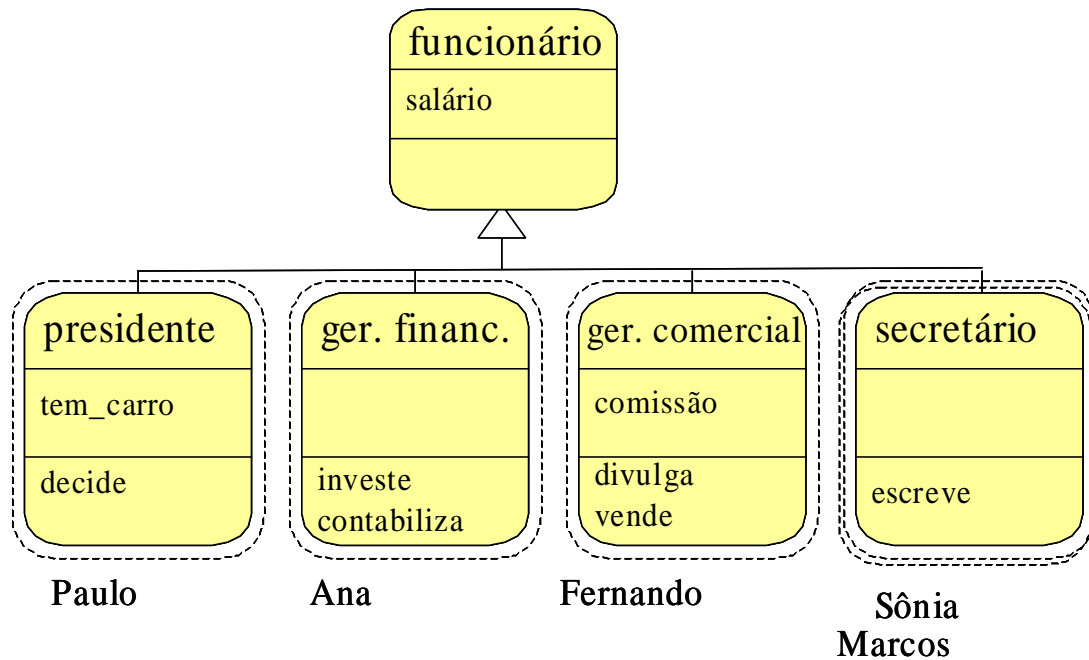


Figura 20: Visão gráfica das entidades (classes) de uma empresa (2)

Em mais um passo na manutenção da classificação do quadro funcional, pode-se imaginar o crescimento da empresa e a necessidade de se criar uma nova classe que seria um gerente financeiro *treinee*, chamado Mauro, que faz o que o gerente financeiro faz e ainda estuda. O nosso quadro de classificação ficará assim:

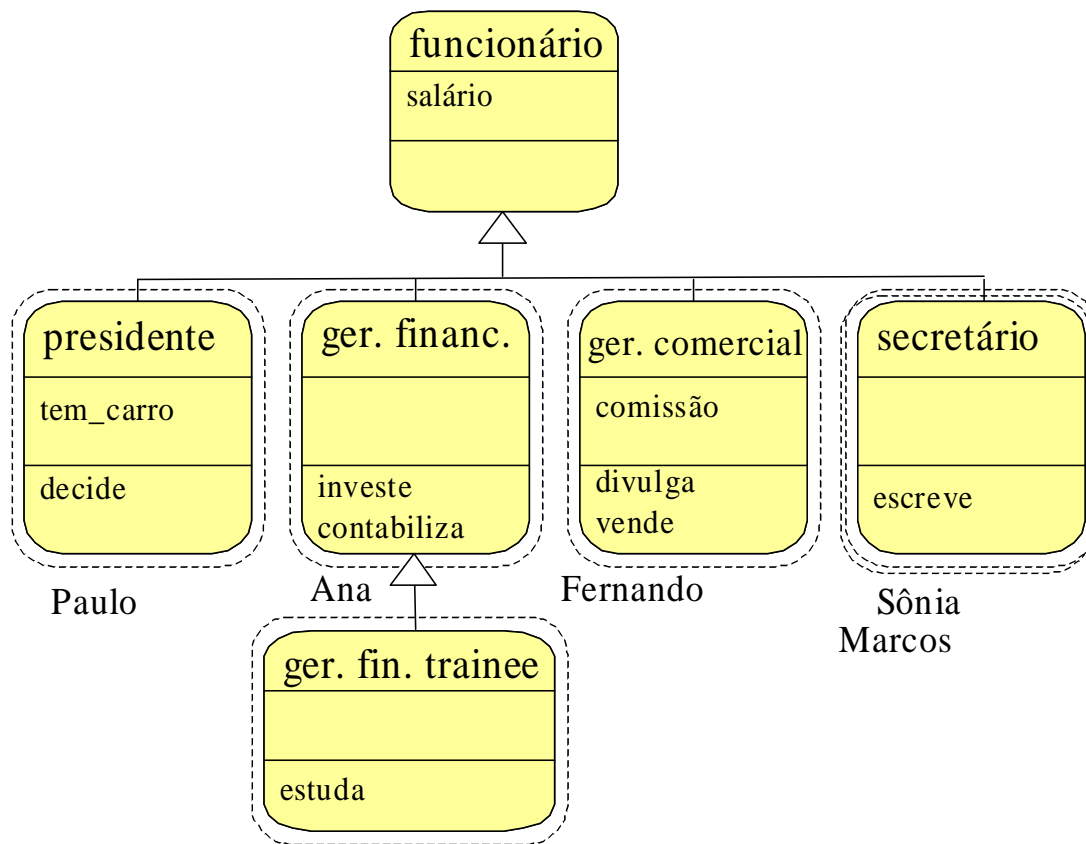


Figura 21: Visão gráfica das entidades (classes) de uma empresa (3)

9.8 Herança

A herança é o processo pelo qual um objeto adquire (herda) propriedades de outro, conforme foi analisado conceitualmente no caso da empresa. Esse conceito é muito importante pois abarca a noção de classificação. No exemplo a seguir, uma outra situação é abordada, sendo a codificação em C++ exibida em relacionamento direto. Um fusca é um automóvel (*automobile*) e um automóvel é um veículo (*vehicle*); um caminhão (*truck*) também é um veículo e um FNM é um caminhão. Sem a classificação e a herança, cada classe (as palavras sublinhadas) teria de ser definido por todas as suas propriedades. Digamos que cada uma das 3 classes definidas possua um atributo e 2 métodos. O automóvel possui um atributo que é “rodas” (número de rodas), que em inglês é *wheels*. Como trata-se de um dado membro, o identificador desse atributo é *m_wheels*. Há dois métodos na classe veículo, um para definir o número de rodas e outro para obter o número de rodas. A classe caminhão possui um atributo que é a carga (que pode transportar). O identificador desse atributo é *m_carga*. Há dois métodos na classe caminhão – um para definir a carga e outro para obter a carga. A classe automóvel possui um atributo cujo identificador é *m_type*, que identifica o automóvel como do tipo *trail*, *city* ou *city_luxury* (trilha, cidade ou

cidade_luxo), e dois métodos para definir e obter o tipo do automóvel. Fusca e FNM são objetos, isto é, ocorrências (instâncias) de automóvel e caminhão, respectivamente. Vejamos um diagrama de classificação para este caso.

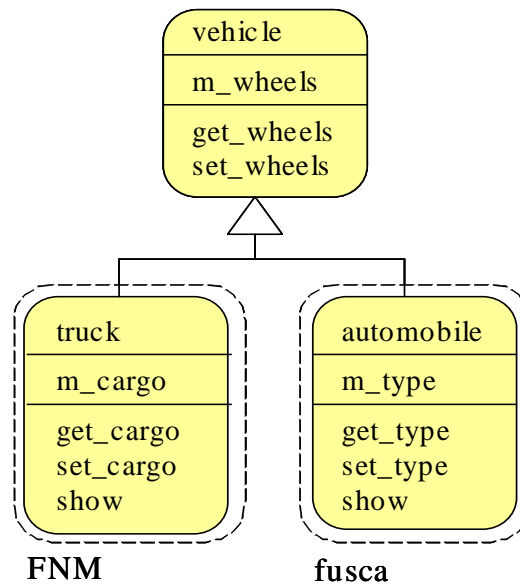


Figura 22: Visão gráfica das entidades (classes) para carros

Vejamos agora uma implementação disto em C++:

```

#include <iostream.h>
// the base class and its base properties
class vehicle
{
    int m_wheels;
public:
    void set_wheels(int wheels)
    {
        m_wheels = wheels;
    };
    int get_wheels()
    {
        return m_wheels;
    };
};

// the first derived class and its specific additional properties
class truck : public vehicle
{
    int m_cargo;
public:
    void set_cargo(int cargo)
    {
        m_cargo = cargo;
    };
    int get_cargo()
    {
        return m_cargo;
    };
    void show()

```

```

        {
            cout << "wheels = " << get_wheels() << endl
                 << "carga = " << get_carga() << endl;
        };
};

// the second derived class and its specific additional properties
enum a_type {trail, city, city_luxury};
class automobile : public vehicle
{
    a_type m_auto_type;
public:

    void set_type(a_type auto_type)
    {
        m_auto_type = auto_type;
    };
    a_type get_type()
    {
        return m_auto_type;
    };
    void show()
    {
        cout << "wheels = " << get_wheels() << endl
             << "type = " << get_type() << endl;
    };
};

int main()
{
    truck FNM;
    automobile fusca;
    FNM.set_wheels(18);
    FNM.set_carga(3200);
    FNM.show();
    fusca.set_wheels(4);
    fusca.set_type(city);
    fusca.show();
    return 0;
}
Resultado:
wheels = 18
carga = 3200
wheels = 4
type = 1

```

Suponha que depois de uma versão pronta do programa, surja necessidade de upgrade devido a mudança na especificação de requisitos. Agora é preciso definir o tipo de freio (*breaks*) e o volume do motor do veículo. Em princípio se poderia acrescentar tais características como novos atributos da classe veículo, e elas passariam a pertencer também às classes caminhão e automóvel. Mas poderia ser também que seu quadro de classificação já possuísse as classes freio e motor e que você quisesse aproveitá-las. Ou então pode-se prever que no futuro alguma necessidade poderá requerer características de freio ou de motor dissociadas das características de veículo. Neste caso pode ser mais interessante criar novas classes de freio e motor, e a classe veículo herdaria de mais de uma classe, o que se chama de herança múltipla.

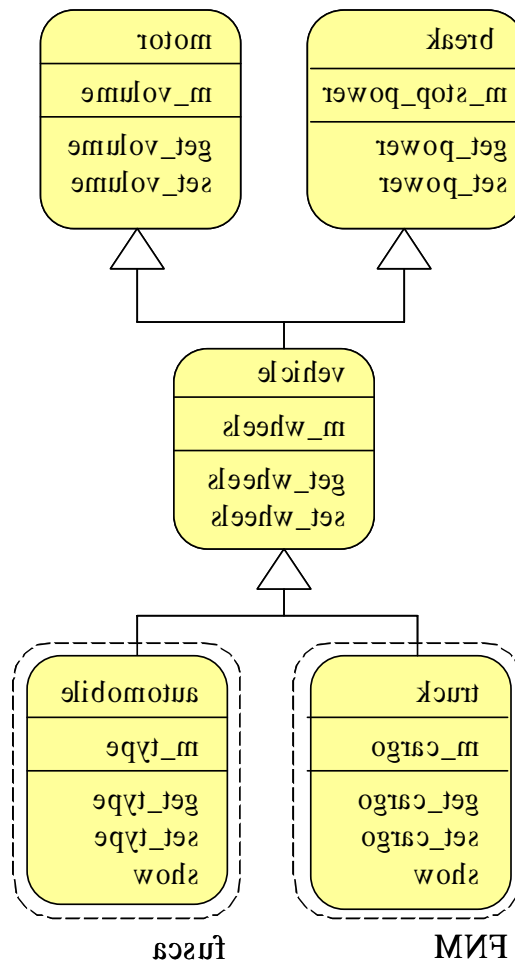


Figura 23: Visão gráfica das entidades (classes) para carros (2)

Para implementar isto em C++ a partir do código anterior, basta definir as novas classes e fazer a classe veículo herdar delas. Nesse caso, os métodos show foram alterados para mostrar também as novas características das classes caminhão e automóvel.

```

#include <iostream.h>
class breaks
{
    float m_stop_power;
public:
    void set_spower(float stop_power)
    {
        m_stop_power = stop_power;
    };
    float get_spower()
    {
        return m_stop_power;
    };
};

class motor
{

```

```

        float m_volume;
public:
    void set_volume(float volume)
    {
        m_volume = volume;
    };
    float get_volume()
    {
        return m_volume;
    };
};

// the base class and its base properties
class vehicle : public breaks, public motor
{
    int m_wheels;
public:
    void set_wheels(int wheels)
    {
        m_wheels = wheels;
    };
    int get_wheels()
    {
        return m_wheels;
    };
};

// the first derived class and its specific additional properties
class truck : public vehicle
{
    int m_cargo;
public:
    void set_cargo(int cargo)
    {
        m_cargo = cargo;
    };
    int get_cargo()
    {
        return m_cargo;
    };
    void show()
    {
        cout << "wheels = " << get_wheels() << endl
              << "stop power = " << get_spower() << endl
              << "motor volume = " << get_volume() << endl
              << "cargo = " << get_cargo() << endl;
    };
};

// the second derived class and its specific additional properties
enum a_type {trail, city, city_luxury};
class automobile : public vehicle
{
    a_type m_auto_type;
public:
    void set_type(a_type auto_type)
    {
        m_auto_type = auto_type;
    };
    a_type get_type()
    {

```

```

        return m_auto_type;
    };
    void show()
    {
        cout << "wheels = " << get_wheels() << endl
              << "stop power = " << get_spower() << endl
              << "motor volume = " << get_volume() << endl
              << "type = " << get_type() << endl;
    };
};

int main()
{
    truck FNM;
    automobile fusca;
    FNM.set_wheels(18);
    FNM.set_spower(333.3);
    FNM.set_volume(5.5);
    FNM.set_cargo(3200);
    FNM.show();
    fusca.set_wheels(4);
    fusca.set_spower(100.2);
    fusca.set_volume(1.5);
    fusca.set_type(city);
    fusca.show();
}

```

Resultado:

```

wheels = 18
stop power = 333.3
motor volume = 5.5
cargo = 3200
wheels = 4
stop power = 100.2
motor volume = 1.5
type = 1

```

Pode-se argumentar com razão que o método proposto ainda não está bom o suficiente. Isso porque a alteração das características da classe veículo obrigou mudança nas classes derivadas. Além disso, há código repedito nos métodos show das classes automóvel e caminhão. Isso prejudica a manutibilidade. Uma forma de se melhorar a manutibilidade é re-escrever o programa acima fazendo as classes automóvel e caminhão implementarem o método show chamando um método show da classe veículo e acrescentando apenas suas características próprias. Com isso, por alterar o método show da classe veículo se estará alterando o método show das classes derivadas (automóvel e caminhão). Caso se defina o mesmo nome (show) para o método na classe veículo, então será preciso uma sintaxe própria para uma classe derivada chamar o método de mesmo nome de sua classe base. Isso pode ser feito com o operador de escopo (::) [vehicle::show()].

Na nova versão do programa mostrada abaixo, com comportamento idêntico, há um método show na classe veículo, que é chamado pelo método show das classes automóvel e caminhão.

```

#include <iostream.h>
class breaks

```



```

{
    float m_stop_power;
public:
    void set_spower(float stop_power)
    {
        m_stop_power = stop_power;
    };
    float get_spower()
    {
        return m_stop_power;
    };
};

class motor
{
    float m_volume;
public:
    void set_volume(float volume)
    {
        m_volume = volume;
    };
    float get_volume()
    {
        return m_volume;
    };
};

// the base class and its base properties
class vehicle : public breaks, public motor
{
    int m_wheels;
public:
    void set_wheels(int wheels)
    {
        m_wheels = wheels;
    };
    int get_wheels()
    {
        return m_wheels;
    };
    void show()
    {
        cout << "wheels = " << get_wheels() << endl
              << "stop power = " << get_spower() << endl
              << "motor volume = " << get_volume() << endl;
    };
};

// the first derived class and its specific additional properties
class truck : public vehicle
{
    int m_cargo;
public:
    void set_cargo(int cargo)
    {
        m_cargo = cargo;
    };
    int get_cargo()
    {
        return m_cargo;
    };
    void show()

```

```

        {
            vehicle::show();
            cout << "cargo = " << get_cargo() << endl;
        };
};

// the second derived class and its specific additional properties
enum a_type {trail, city, city_luxury};
class automobile : public vehicle
{
    a_type m_auto_type;
public:

    void set_type(a_type auto_type)
    {
        m_auto_type = auto_type;
    };
    a_type get_type()
    {
        return m_auto_type;
    };
    void show()
    {
        vehicle::show();
        cout << "type = " << get_type() << endl;
    };
};

int main()
{
    truck FNM;
    automobile fusca;
    FNM.set_wheels(18);
    FNM.set_spower(333.3);
    FNM.set_volume(5.5);
    FNM.set_cargo(3200);
    FNM.show();
    fusca.set_wheels(4);
    fusca.set_spower(100.2);
    fusca.set_volume(1.5);
    fusca.set_type(city);
    fusca.show();
    return 0;
}

```

9.9 Herança múltipla e classe base virtual

Se uma classe possui herança múltipla, é possível que existam múltiplas cópias de um mesmo atributo herdado. Isso pode ocorrer no caso de uma classe derivada em nível 2 com heranças múltiplas de duas ou mais classes que herdem de uma mesma classe base. Nesse caso, os atributos da classe base aparecem mais de uma vez na classe derivada em nível 2. Para solucionar esse problema é preciso que se introduza o conceito de *classe base virtual*. O suporte a herança múltipla não é implementado em todas as linguagens OO. Java, por exemplo, não suporta herança múltipla. Ou melhor, permite herança múltipla desde que no máximo uma classe seja não-abstrata e as demais sejam classes abstratas.

Em C++, pode ocorrer herança múltipla com mais de uma classe base não abstrata.

Seja o diagrama de herança de classes da figura abaixo. A classe `derived_level_2` possui herança múltipla de duas classes não abstratas – `derived_A` e `derived_B`. O diagrama mostra que essas duas classes herdam (herança simples) da classe base.

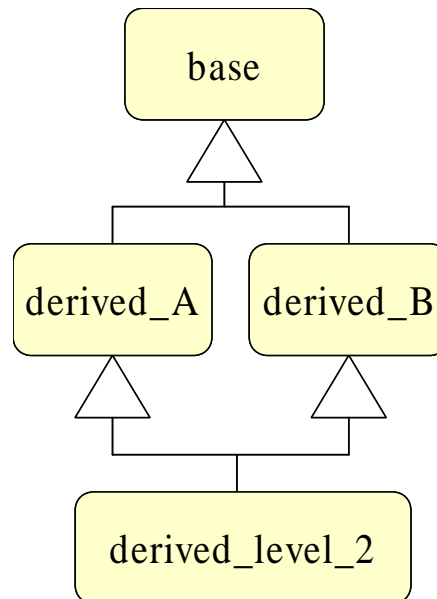


Figura 24: Exemplo de classe com duas cópias de um atributo herdado devido a herança múltipla

Em princípio um objeto da classe `derived_level_2` possuirá 2 vezes o conteúdo da base, uma vez em função da herança de `derived_A`, e outra vez em função da herança de `derived_B`. A tentativa de acessar diretamente um atributo ou método da classe base pelo objeto da classe `derived_level_2` gera um erro de ambigüidade, a menos que seja feita referência explicitamente sobre de qual das duas classes derivadas nível 1 se herdou o membro da base. Mas raramente tal recurso é útil. É mais provável que se queira apenas 1 cópia da base na classe `derived_level_2`, e para tanto será necessário definir nas classes derivadas de nível 1 (`derived_A` e `derived_B`) uma herança virtual da classe base. Desta forma o compilador sabe que antes de criar qualquer herdeiro desta classe, os membros da classe base poderão ser fundidos de modo a criar apenas 1 cópia na classe herdeira.

Veja o exemplo (o programa compila mas não gera resultados na tela). Nesse exemplo, são definidas 7 classes, com diagrama de herança como mostrado na figura abaixo.

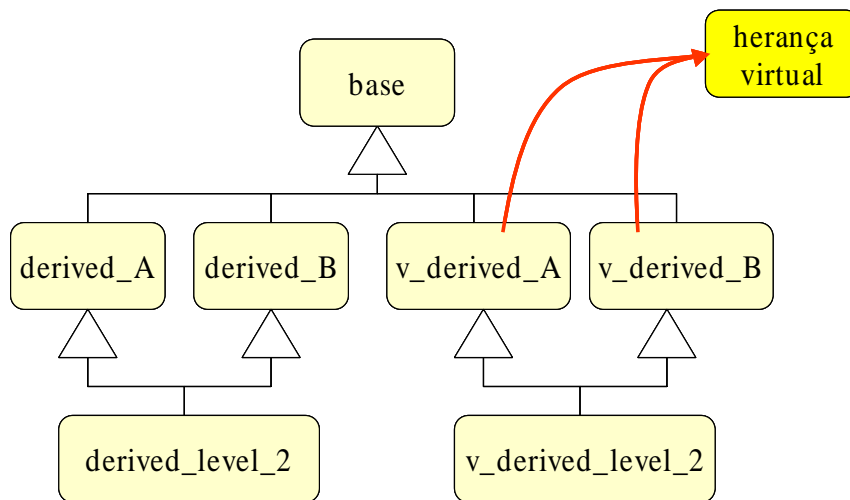


Figura 25: Com herança virtual, mesmo havendo herança múltipla não ocorre multiplicidade de atributo herdado

Devido ao problema exposto, a classe `derived_level_2` possui duas cópias do atributo `m_base_i` (da classe `base`). Um dos atributos foi herdado de `derived_A` e o outro de `derived_B`. O objeto “a”, da classe `derived_level_2`, precisa explicitar de qual das cópias de `m_base_i` se refere (pois do contrário causa um erro de ambigüidade).

Já a classe `v_derived_level_2` possui apenas uma cópia do atributo `m_base_i`, pois embora herde de `v_derived_A` e de `v_derived_B`, essas duas classes herdam de forma virtual da classe `base`. Portanto, o objeto “v” da classe `v_derived_level_2` pode acessar o atributo `m_base_i` diretamente. Há apenas uma cópia de `m_base_i` no objeto “v”.

```

class base
{
public:
    int m_base_i;
};

class derived_A : public base
{
public:
    int m_derived_A_i;
};

class derived_B : public base
{
public:
    int m_derived_B_i;
};

class derived_level_2 : public derived_A, public derived_B
{
public:
    int m_derived_level_2_i;
};
  
```

```

// inherits from base as virtual
class v_derived_A : virtual public base
{
public:
    int m_derived_A_i;
};

// inherits from base as virtual
class v_derived_B : virtual public base
{
public:
    int m_derived_B_i;
};

class v_derived_level_2 : public v_derived_A, public v_derived_B
{
public:
    int m_derived_level_2_i;
};

int main()
{
    derived_level_2 a;
    v_derived_level_2 v;

    // fill all attributes of object "a"
    a.derived_A::m_base_i = 1;
    a.derived_B::m_base_i = 5;
    a.m_derived_A_i = 2;
    a.m_derived_B_i = 3;
    a.m_derived_level_2_i = 4;

    // fill all attributes of object "v"
    v.m_base_i = 1;
    v.m_derived_A_i = 2;
    v.m_derived_B_i = 3;
    v.m_derived_level_2_i = 4;
    return 0;
}

```

9.10 União adiantada × união tardia (*early bind* × *late bind*)

Um ponteiro para uma classe base pode apontar objetos de classes derivadas. O contrário não pode ser feito (um ponteiro para classe derivada apontar para um objeto de uma classe base). Isso ocorre porque a definição de herança é que um objeto de classe derivada possui todos os atributos e métodos da classe base. O ponteiro para classe base “saberá” apontar para os atributos e métodos da classe derivada, embora não “não saiba” apontar para os atributos e métodos da classe derivada.

```

class base
{
public:
    int i;
    void fun();
};

class derived : public base
{

```

```

public:
    double d;
    void fun2();
};

int main()
{
    base* pBase;
    derived derivedObj;
    pBase = & derivedObj;
    pBase->i = 1;
    pBase->fun();
    pBase->d; // erro: d pertence a classe derivada e não a classe base
    pBase->fun2(); // erro: fun() pertence a classe derivada e não a classe base
    derivedObj.d = 2.2; // OK
    derivedObj.fun2(); // OK
    return 0;
}

```

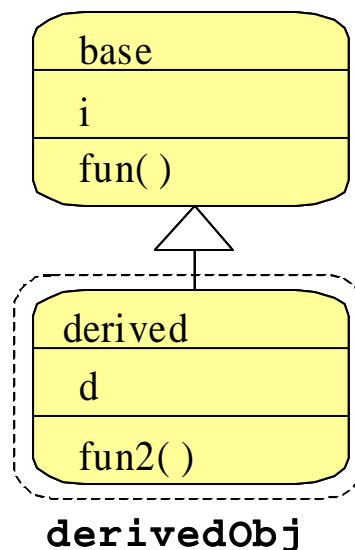


Figura 26: Diagrama de herança de classe base e derivada, ilustrando a união tardia

Seja o caso em que a classe base e classe derivada possuem um método de mesmo nome. No exemplo abaixo, o método chama-se show.

```

#include <iostream.h>
class base
{
public:
    void show() { cout << "base class" << endl; }
};

class derived : public base
{
public:
    void show()
    {
        cout << "derived class" << endl;
    }
};

```

```
int main()
{
    base *pBase;
    derived derivedObj;
    pBase = &derivedObj; // late bind
    pBase->show();
    derivedObj.show();
    return 0;
}
```

Resultado

```
base class
derived class
```

No caso do exemplo acima, a união do ponteiro base com a classe derivada é chamado de “união adiantada” (*early bind*). O tempo adiantado (*early*) se refere ao fato de que a união ocorre em tempo de compilação. Mas é possível um outro tipo de “união tardia” (*late bind*), que ocorre em tempo de execução. Em C++ programa-se a união tardia com a palavra reservada “virtual”. Considere agora o mesmo código acima, apenas acrescentando a palavra “virtual” no local indicado na classe base.

```
class base {
public:
    virtual void show() { cout << "base class" << endl; }
};
```

Agora o resultado é:

```
derived class
derived class
```

Isso significa que embora o ponteiro pBase aponte para a classe base, como o método em questão “show” foi programado como virtual, o ponteiro para esse método, em tempo de execução (i.e. união tardia), recebeu o endereço do método da correspondente da classe derivada.

A inclusão da palavra reservada “virtual” antes de um método torna-o elegível para ser sujeito de uma união tardia. Isto é, se um método é “marcado” com virtual, isso significa que caso ocorra a união de um ponteiro para classe base recebendo o endereço de um objeto derivado, o compilador procurará por um método como mesmo identificador, e se encontrar, em tempo de execução, o ponteiro será carregado com o valor que o faz apontar para o método da classe derivada.

9.10.1 Classe abstrata

Pode-se definir um método numa classe base sem implementação, definido com a única finalidade de se unir tardiamente. Nesse caso o método é dito “puramente virtual”. Uma classe que possua pelo menos um método puramente virtual é chamada de “classe abstrata”. Não pode haver um objeto que seja uma ocorrência (instância) de uma classe abstrata.

No programa abaixo, a classe “base” é abstrata, e o método “show” é puramente virtual.

```
#include <iostream.h>
class base
{
public:
    virtual void show() = 0; // puramente virtual
};

class derived : public base
{
public:
    void show()
    {
        cout << "derived class" << endl;
    }
};

int main()
{
    base *pBase;
    derived derivedObj;
    pBase = &derivedObj; // late bind
    pBase->show();
    derivedObj.show();
    base baseObj; // erro: não pode haver objeto de classe abstrata
    return 0;
}
```

O resultado é:

```
derived class
derived class
```

Abaixo é mostrado um exemplo simples de programa que usa uma classe abstrata chamada “figure”. Existem 3 classes não abstratas que herdam de figure – triangle, square e circle (triângulo, quadrado e círculo). A figura abaixo ilustra o diagrama de heranças dessas classes.

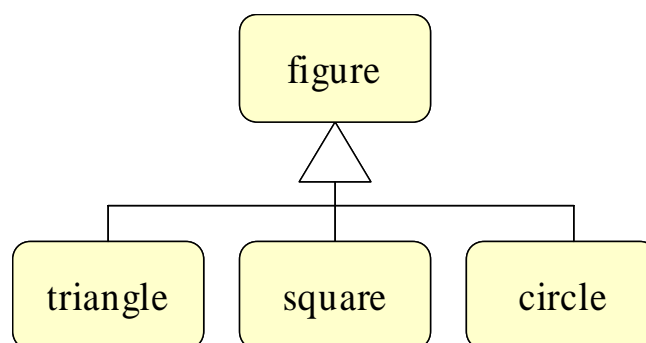


Figura 27: Diagrama de herança das classes do exemplo

Nesse exemplo, pergunta-se pelo console qual a figura que se deseja. Desde que a resposta será digitada pelo usuário, não se pode saber em tempo de compilação a resposta da pergunta. Para qualquer resposta válida que se entre, o programa fará a união tardia do ponteiro da classe figure com a classe derivada

correspondente. Em seguida, o programa executa o método show a partir da classe abstrata. Ao usar esse método, se está chamando o método show da classe correspondente sem saber (em tempo de compilação) que classe é essa.

No exemplo abaixo, a união tardia ocorre quando se aloca um objeto da classe derivada e armazena-se em um ponteiro para classe base. Para se deletar o objeto alocado, usa-se o ponteiro da classe base.

```
#include <iostream.h>
class figure
{ // abstract base class
public:
    virtual void show() = 0; // pure virtual
};
class square : public figure
{
public:
    void show()
    {
        cout << "square" << endl;
    };
};
class circle : public figure
{
public:
    void show()
    {
        cout << "circle" << endl;
    };
};
class triangle : public figure
{
public:
    void show()
    {
        cout << "triangle" << endl;
    };
};
figure *enterFigure()
{
    figure *ret = 0;
    int figNum;
    cout << "1 = square, 2 = circle, 3 = triangle" << endl;
    cout << "Enter a figure: ";
    cin >> figNum;
    switch (figNum)
    {
        case 1:
            ret = new square;
            break;
        case 2:
            ret = new circle;
            break;
        case 3:
            ret = new triangle;
            break;
        default:
            cout << "Figure not recognized" << endl;
            ret = 0;
            break;
    }
}
```

```

    };
    return ret;
}
int main()
{
    figure *f;
    f = enterFigure();
    if (f)
    {
        f->show(); // call show method of the class chosen by user
        delete f;
    }
    return 0;
}

```

O resultado é:

```

1 = square, 2 = circle, 3 = triangle
Enter a figure:2↵
circle

```

O uso (chamada) do método show é feito a partir do ponteiro para classe base. Essa parte do desenvolvimento do software é feita de forma isolada da implementação do método show, que é feito pelas classes derivadas. Em outras palavras: o código abaixo corresponde a uso de um método sem que se saiba como é implementado. Usa-lo dessa forma é estar isolado da implementação, isto é, pode-se mudar a implementação do método show sem se modificar essa parte do código.

```
f->show();
```

Uma versão desse programa foi adaptado para Windows, usando Visual C++. Copie o programa (da página web do livro) e examine os fontes. Nesse programa, escolhe-se a forma (círculo, quadrado ou triângulo), e para a forma que se escolheu comanda-se a cor do fundo e a cor da linha. Os comandos de cor (de fundo e de linha) são feitos pelo ponteiro para classe base abstrata.

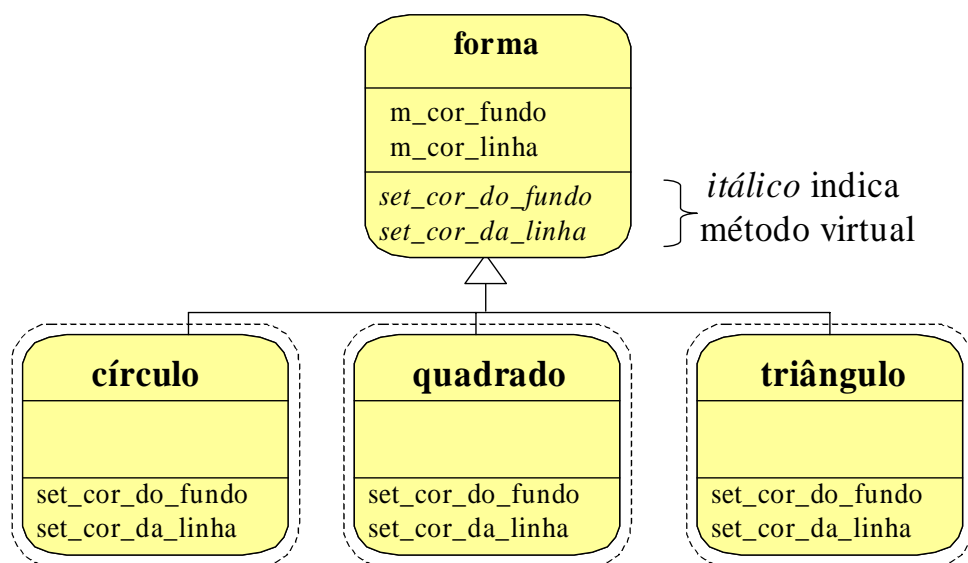


Figura 28: Diagrama de herança de classes ilustrando uso de classe abstrata

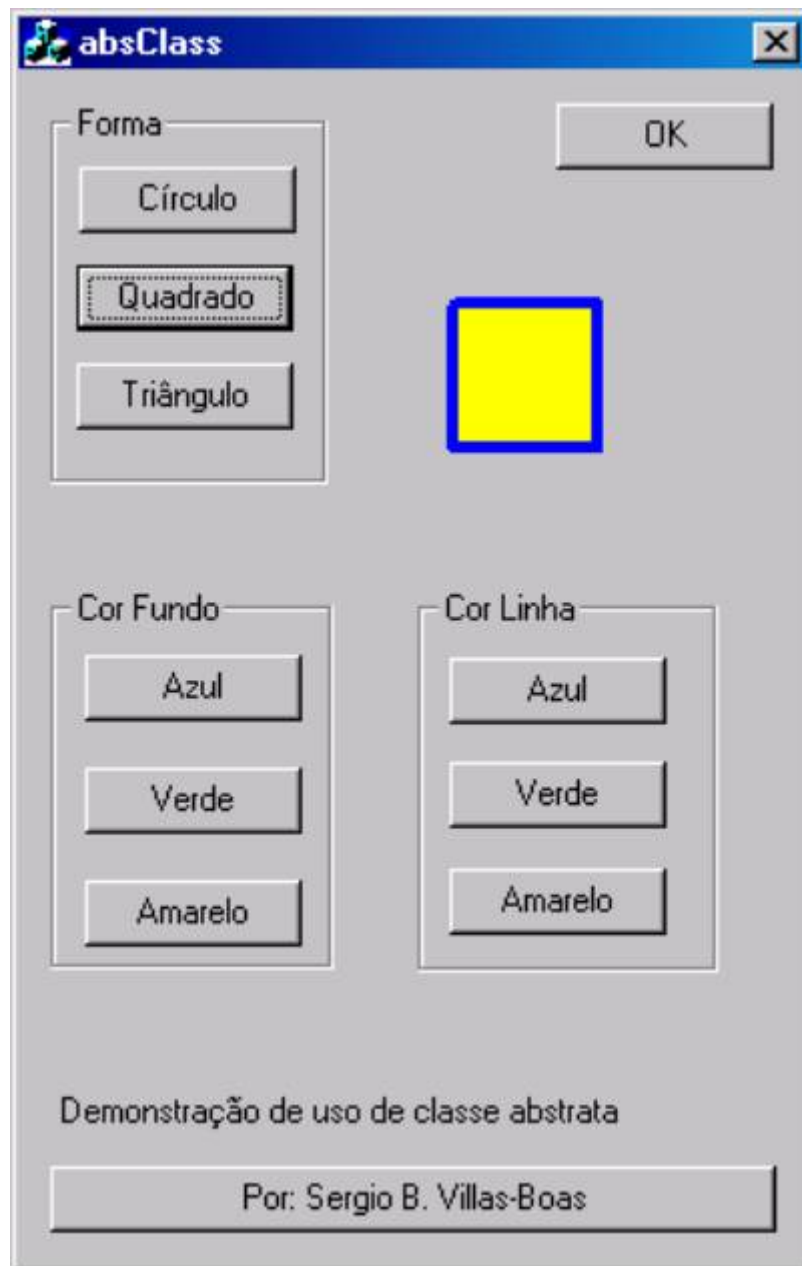


Figura 29: Caixa de diálogo do programa para Windows de Exemplo de uso de classe abstrata

9.10.2 Porque usar união tardia ?

O uso de união tardia é um poderoso recurso de programação OO. O seu uso pode trazer grande flexibilidade e manutibilidade ao software desenvolvido. Por usar união tardia, se está isolando a utilização de métodos da implementação dos mesmos.

Imagine por exemplo que se está desenvolvendo um jogo. Trata-se de um jogo em que o usuário escolhe o personagem de uma lista, e entra no cenário. Um

software assim pode ser desenvolvido com grande benefício do uso de união tardia. Seja uma classe abstrata “personagem”, que contém os métodos que se pretende dar suporte. Os métodos virtuais podem ser “andar para esquerda, andar para direita, andar para frente, andar para trás, pular, aplicar golpe 1, aplicar golpe 2, etc.”. Na primeira versão do software, desenvolve-se 4 personagens – person_1 ~ person_4 – que herdam de “personagem”. Cada um deles implementa de forma diferente os métodos virtuais. O usuário escolhe o personagem que quiser e joga o jogo observando a implementação dos métodos da classe que escolheu. Mas o jogo em si (suporte do teclado ou joystick para os métodos virtuais, eventos aleatórios do jogo, etc.) é todo desenvolvido sem que se saiba qual personagem o usuário escolheu. O desenvolvimento do jogo (uso dos métodos) é isolado do problema de implementação dos métodos.

Por utilizar união tardia, pode-se paralelizar o desenvolvimento do jogo. Enquanto uma equipe se concentra em desenvolver o jogo em si, n outras pessoas podem se concentrar em desenvolver classes com personagens para compor o jogo. Além disso, por usar união tardia fica-se em muito boa situação para se desenvolver novas versões do jogo com novos personagens. O acréscimo de novas classes com personagens não requer modificação de nenhuma outra parte do código.

9.11 Construtor e destrutor de um objeto

Os objetos, quando são definidos, podem ser automaticamente inicializados por métodos conhecidos como construtores. Geralmente cabe a esses métodos a responsabilidade de inicializar atributos, alocar memória, abrir arquivos, etc. O construtor de uma classe é um método que tem o mesmo nome da própria classe. O construtor é um método da classe que, ao contrário dos demais métodos, não é chamado explicitamente pelo programador, mas implicitamente pelo compilador. Por isso, não faz sentido o retorno do método construtor, e portanto não se escreve nem mesmo void como retorno. Análogo ao método construtor, existe o método destrutor, que possui o mesmo nome do construtor (e da classe) precedido por ~ (letra til). O destrutor também não retorna. O construtor é chamado quando o objeto é construído (instanciado ou alocado com new), e o destrutor é chamado quando o objeto sai do escopo ou é deletado com delete.

No exemplo abaixo, a classe myClass possui construtor (implícito) e destrutor, que sinalizam para o console. Há dois objetos dessa classe “a” e “b”. Repare que o escopo do objeto “b” acaba antes do escopo do objeto “a”, portanto o destrutor do objeto “b” é chamado antes do destrutor do objeto “a”.

```
#include <iostream.h>
class myClass
{
public:
    myClass() // default constructor
```

```

    {
        cout << "constructor" << endl;
    }
    ~myClass() // destructor
    {
        cout << "destructor" << endl;
    }
};

int main() {
    myClass a;
    {
        myClass b;
    }
    return 0;
}

```

O programa acima dá o seguinte resultado no console (a marcação de qual construtor/destrutor pertence a qual objeto aparece apenas como referência).

```

constructor (a)
constructor (b)
destructor (b)
destructor (a)

```

Uma mesma classe pode possuir mais de um construtor. Apesar de o construtor necessariamente ter o nome da classe, pode ocorrer mais de um construtor pelo princípio de polimorfismo. O construtor sem parâmetros é conhecido como construtor implícito (*default constructor*). Esse é o caso do esemplo mostrado acima. Quase todas as classes que tem construtor tem construtor implícito, mas pode ocorrer de uma classe ter construtor não implícito e não ter construtor implícito. Somente pode haver destrutor sem parâmetros, por isso nunca ocorre mais de um destrutor para a mesma classe.

No exemplo abaixo, a classe myClass possui 3 construtores e um destrutor. Repare na sintaxe para instanciar um objeto, passando parâmetros pelo construtor.

```

#include <iostream.h>
class myClass
{
    double m_d;
    int m_i;
public:
    myClass();           // Construtor sem parametros (default constructor)
    myClass(int);        // Construtor com um parâmetro
    myClass(int, double); // Construtor com dois parâmetro
    ~myClass();          // Destrutor
};

myClass::myClass()
{
    cout << "Construtor sem parâmetros" << endl;
}

myClass::myClass(int i)
{
    cout << "Construtor com 1 parâmetro" << endl;
    m_i = i;
}

myClass::myClass(int i, double d)

```

```

{
    cout << "Construtor com 2 parâmetros" << endl;
    m_i = i;
    m_d = d;
}
myClass::~myClass()
{
    cout << "destrutor" << endl;
}
void m()
{
    myClass f1;
    myClass f2(1);
    myClass f3(1, 3.14);
}
int main()
{
    m();
    return 0;
}

```

Resultado:

```

Construtor sem parâmetros
Construtor com 1 parâmetro inteiro
Construtor com 1 parâmetro inteiro e um double
destrutor
destrutor
destrutor

```

9.11.1 Default constructor (construtor implícito)

Um construtor especial que existe num objeto é o default constructor, que é o construtor que não tem parâmetros. Se uma classe não define explicitamente construtor algum o compilador cria um construtor implícito (isto é, sem parâmetros), que não faz nada. Algo como mostrado abaixo.

```
myClass() {} // empty default constructor
```

O default constructor é o construtor chamado quando se declara um objeto sem nenhum parâmetro no construtor, como é muito usual (mostrado abaixo).

```

void f() {
    myClass myObj; // default constructor called
}

```

Caso uma classe declare um construtor implícito, o construtor declarado é usado no lugar do construtor vazio que o compilador usaria. Mas atenção: caso a classe tenha um construtor com parâmetros e não declare explicitamente um construtor implícito, então torna-se erro de compilação declarar um objeto sem parâmetros no construtor.

```

class myClass
{
    int m_i;
public:
    myClass(int i) { m_i = i; } // non-default constructor defined
};

void f()
{
    myClass myObj; // error: no appropriate default constructor available
}

```

Esse erro de compilação poderia ser evitado (caso seja conveniente para a lógica de programação em questão) com a introdução de um default constructor explicitamente, como mostrado abaixo.

```
class myClass
{
    int m_i;
public:
    myClass() {}: // explicit definition of default constructor
    myClass(int i) { m_i = i; } // non-default constructor defined
};

void f()
{
    myClass myObj; // OK
}
```

9.11.2 Ordem de chamada de construtor e destrutor para classes derivadas

Caso haja uma classe derivada de uma classe base, sendo ambas com construtor e destrutor, como é de se esperar, será chamado o construtor e o destrutor tanto da classe base quanto da classe derivada. Fato semelhante ocorre no caso de uma classe possuir um atributo que é um objeto de outra classe.

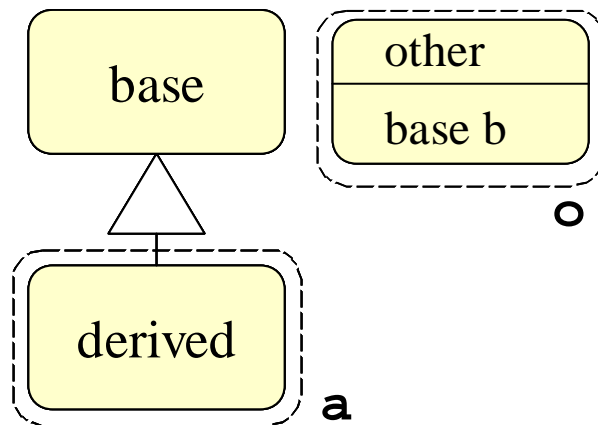


Figura 30: relacionamento entre classes: herança e posse de atributo

No exemplo abaixo, há 3 classes – base, derived e other. Derived herda de base, e há um objeto “a” dessa classe. Other não herda de ninguém, mas possui um atributo do tipo base. Há um objeto “o” do tipo other. Como todas as classes possuem construtor e destrutor, e sinalizam isso no console, o programa abaixo ilustra a ordem de chamada de construtor e destrutor.

```
#include <iostream.h>
class base
{
public:
    base() { cout << "constructor base" << endl; }
    ~base() { cout << "destructor base" << endl; }
};
```

```

class derived : public base
{
public:
    derived() { cout << "constructor derived" << endl; }
    ~derived() { cout << "destructor derived" << endl; }
};

class other
{
public:
    other() { cout << "constructor other" << endl; }
    ~other() { cout << "destructor other" << endl; }
    base b;
};

int main()
{
    derived a;
    cout << "----" << endl;
    other o;
    cout << "----" << endl;
    return 0;
}

```

A saída do programa é como mostrado abaixo.

```

constructor base
constructor derived
---
constructor base
constructor other
---
destructor other
destructor base
destructor derived
destructor base

```

Seja a variante do programa anterior, em que se cria um escopo para o objeto “a” da classe derived e para o objeto “o”, da classe other. Apenas a função main foi re-escrita como mostrado abaixo.

```

int main()
{
    {
        derived a;
    }
    cout << "----" << endl;
    {
        other o;
    }
    cout << "----" << endl;
    return 0;
}

```

Nesse caso a saída do programa é diferente, como mostrado abaixo. A diferença ocorre porque o fim do escopo dos objetos faz chamar o destrutor.

```

constructor base
constructor derived
destructor derived
destructor base
---
constructor base

```



```

constructor other
destructor other
destructor base
---
```

9.11.3 Inicialização de atributos com construtor não implícito

Desde que uma classe pode ter mais de um construtor (usando polimorfismo) pode-se querer controlar qual construtor uma classe derivada chamará. Implicitamente o compilador tentará sempre usar o construtor sem parâmetro (*default constructor*); para forçar a chamada de um outro construtor (com parâmetro), basta declara-lo na sequência do construtor da classe derivada separado por “dois pontos” (“:”).

No exemplo abaixo, uma classe base e uma classe derivada possuem construtor padrão e não padrão. O construtor não padrão da classe derivada chama o construtor não padrão da classe base.

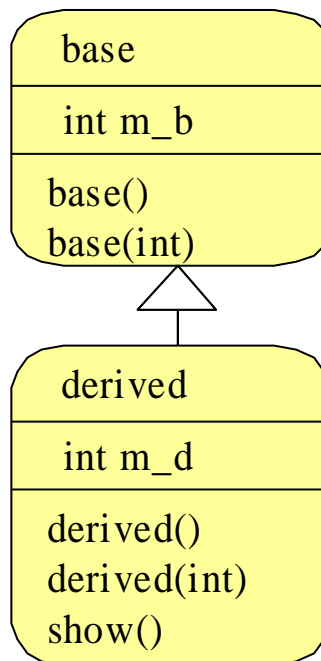


Figura 31: classe base e derivada, com construtor não padrão

```

#include <iostream.h>
class base
{
public:
    int m_b;
    // default constructor
    base()
    {
        m_b = 0;
    };
    // constructor with one parameter
    base(int i)
    {
        m_b = i;
```

```

    };
}; // end of base class
class derived : public base
{
    int m_d;
public:
    // default constructor
    derived()
    {
        m_d = 0;
    };
    // constructor with one parameter calls non-default constructor of base class
    derived(int h) : base(h)
    {
        m_d = h;
    };
    void show()
    {
        cout << "m_b = " << m_b << ", m_d = " << m_d << endl;
    };
};

int main()
{
    derived d1;
    derived d2(2);
    d1.show();
    d2.show();
    return 0;
}

```

A saída do programa é como mostrado abaixo. Repare que no caso do objeto d2, passa-se parâmetro para o construtor (da classe derivada). Mas o atributo da classe base `m_b` foi inicializado com valor diferente de zero, mostrando que o construtor não padrão da classe base foi chamado nesse caso.

```

m_b = 0, m_d = 0
m_b = 2, m_d = 2

```

No caso de uma classe que não herda propriedades de uma classe base, mas que possua atributos de uma classe que tenha construtor não padrão, de forma análoga ao caso anterior, é preciso uma sintaxe própria para que se possa inicializar os atributos chamando o construtor não padrão.

No exemplo abaixo, a classe `other` possui dois objetos do tipo base: `m_b1` e `m_b2`. O construtor padrão da classe `other` “não faz nada” (isto é, dentro do escopo do método não há código), mas antes de se abrir o escopo do método construtor da classe `other` faz-se a inicialização dos objetos do tipo base, chamando-se o construtor não padrão desses objetos.

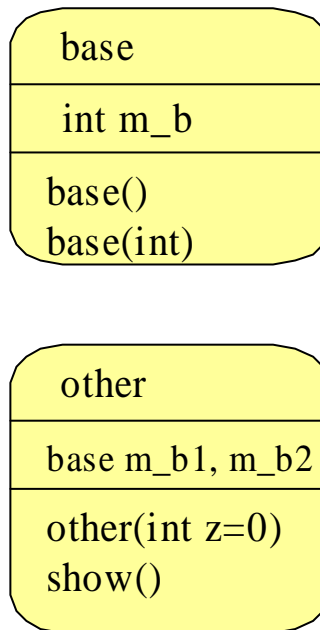


Figura 32: classe other com atributos do tipo base

```
#include <iostream.h>
class base
{
public:
    int m_b;
    // default constructor
    base()
    {
        m_b = 0;
    };
    // constructor with one parameter
    base(int i)
    {
        m_b = i;
    };
}; // end of base class

class other
{
    base m_b1, m_b2;
public:
    // default constructor initializes attributes
    other(int z = 0) : m_b1(z), m_b2(z + 1) {};
    void show()
    {
        cout << "m_b = " << m_b1.m_b << ", m_b2 = " << m_b2.m_b << endl;
    };
};

int main()
{
    other d3;
    other d4(5);
    d3.show();
    d4.show();
    return 0;
}
```

```
}
```

Resultado:

```
m_b = 0, m_b2 = 1
```

```
m_b = 5, m_b2 = 6
```

9.11.4 Construtor de cópia (copy constructor) e operator=

O copy constructor é um construtor que tem o protótipo abaixo.

```
myClass(const myClass & obj); // prototype of copy constructor
```

O copy constructor é chamado automaticamente nas seguintes situações

- Quando um objeto é instanciado e imediatamente atribui-se a ele um valor

```
myClass myObj2 = myObj; // call copy constructor
```

OBS: Quando o objeto é atribuído após a sua instanciação, chama-se o copy constructor e em seguida o método operator=.

- Quando chamado explicitamente

```
myClass myObj3(myObj); // explicit call to copy constructor
```

- Na passagem de um objeto como parâmetro para uma função
- No retorno de um objeto de uma função

Caso uma classe não tenha um construtor de cópia ou operator= definido explicitamente, o compilador cria automaticamente um construtor de cópia ou operator= para ser usado quando necessário. O construtor de cópia ou operator= do compilador copia byte a byte os atributos de um objeto na sua cópia. Esse procedimento resolve muitos problemas mas não todos. Por exemplo: se um objeto aloca memória, o atributo do objeto é o ponteiro para a memória alocada. A cópia byte a byte dos atributos não resulta na cópia da região de memória alocada. Portanto, considerando que cada objeto deve ter sua própria região de memória alocada, é preciso escrever um construtor de cópia e operator= adequado para um objeto que aloca memória.

O código abaixo mostra claramente os momentos em que chama-se o copy constructor, o default constructor, e o operator=. Repare que a cópia da referência (lvalue) ou do ponteiro, como mostrado nas duas últimas linhas do código, não implica na chamada do construtor de cópia nem do operator=.

```
#include <iostream.h>
class myClass
{
public:
    myClass() // default constructor
    {
        static int count = 0;
        cout << "default constructor (" << ++count << ")" << endl;
    };
    myClass(const myClass & obj) // copy constructor
    {
        static int count = 0;
        cout << "copy constructor (" << ++count << ")" << endl;
    }
    void operator= (const myClass & obj)
```

```

    {
        static int count = 0;
        cout << "operator= (" << ++count << ")" << endl;
    };
};

void userFunction(myClass a) // myClass a = myObj; call copy constructor
{ /* do nothing */ }

myClass userFunction2()
{
    myClass a; // default constructor (2)
    return a; // myClass returned_object = a; call copy constructor
}

int main()
{
    myClass myObj; // default constructor (1)
    myClass myObj2 = myObj; // copy constructor (1)
    myObj2 = myObj; // operator= (1)
    myClass myObj3(myObj); // explicit call to copy constructor (2)
    userFunction(myObj); // copy constructor (3)
    myObj2 = // call operator= (2)
    userFunction2(); // default constructor (2), copy constructor (4)
    myClass & l_obj = myObj; // don't call any of these. Reference copy only.
    myClass * p_obj = &myObj; // don't call any of these. Pointer copy only.
    return 0;
}

```

A saída do programa é como mostrado abaixo.

```

default constructor (1)
copy constructor (1)
operator= (1)
copy constructor (2)
copy constructor (3)
default constructor (2)
copy constructor (4)
operator= (2)

```

9.11.5 Destrutores virtuais

Suponha o caso clássico de união tardia (*late bind*). Um ponteiro para classe base aponta para um objeto da classe derivada. De acordo com as regras de união tardia, caso a classe derivada tenha destrutor, esse destrutor somente será chamado caso o destrutor da classe base seja virtual. Por segurança, é aconselhável que os destrutores sejam sempre virtuais. Se uma classe nunca for usada como classe base, a marcação de seu destrutor como virtual ou não é irrelevante.

O exemplo abaixo ilustra o problema. A classe “base” possui destrutor não virtual, e a classe “derived” deriva de “base”. A classe “derived” aloca memória no construtor e apaga a memória no destrutor. Portanto, qualquer objeto da classe derived instanciado diretamente não provoca vazamento de memória. Mas se a classe “derived” for instanciada com união tardia a partir de um ponteiro para classe base (no caso “pb”), então, o destrutor da classe “derived” nunca é chamado. Portanto o programa vaza memória.

A classe “base2” é quase idêntica a “base”, exceto que tem destrutor virtual. A classe “derived2” é quase idêntica a “derived”, exceto que herda de “base2” e não de “base”. O objeto apontado por “pb2”, apesar de ter união tardia, chama corretamente o destrutor da classe “derived2” (além do destrutor da classe “base2”). Essa é a situação segura.

```
#include <iostream.h>
class base
{
public:
    base() { cout << "constructor base" << endl; }
    ~base() { cout << "destructor base" << endl; }
};

class derived : public base
{
    float *m_p;
public:
    derived()
    {
        cout << "constructor derived, allocating of 100 floats" << endl;
        m_p = new float[100];
    }
    ~derived()
    {
        cout << "destructor derived (delete floats)" << endl;
        if (m_p)
        {
            delete[] m_p;
            m_p = 0;
        }
    }
};

class base2
{
public:
    base2() { cout << "constructor base2" << endl; }
    virtual ~base2() { cout << "destructor base2" << endl; }
};

class derived2 : public base2
{
    float *m_p;
public:
    derived2()
    {
        cout << "constructor derived2, allocating of 100 floats" << endl;
        m_p = new float[100];
    }
    ~derived2()
    {
        cout << "destructor derived2 (delete floats)" << endl;
        if (m_p)
        {
            delete[] m_p;
            m_p = 0;
        }
    }
};
```

```

int main()
{
    cout << "--- non virtual destructor" << endl;
    base *pb = new derived;
    delete pb;

    cout << "--- virtual destructor" << endl;
    base2 *pb2 = new derived2;
    delete pb2;
    return 0;
}

```

Esse programa gera a saída como mostrado abaixo.

```

--- non virtual destructor
constructor base
constructor derived, allocating of 100 floats
destructor base
--- virtual destructor
constructor base2
constructor derived2, allocating of 100 floats
destructor derived2 (delete floats)
destructor base2

```

Tanto a classe derived quanto a classe derived2 fazem alocação de memória. Para que a memória não vaze, é preciso que se delete a memória no destrutor. Mas como o objeto da classe derivada está sendo unido ao ponteiro da classe base com união tardia, caso o destrutor não seja virtual, o ponteiro da classe base não se unirá ao destrutor virtual da classe derivada. Nesse caso, o destrutor da classe derivada não é chamado e a memória vaza.

9.11.6 Array de objetos

Para se definir um array estático de objetos, usa-se a mesma sintaxe de um tipo qualquer. O mesmo ocorre quando se quer alocar dinamicamente um array de objetos. Ressalte-se a necessidade de uso de abre e fecha colchete “[]” quando se for deletar o array. O esquecimento do uso do colchete na hora de deletar o array não causa erro no compilador, mas é um bug no programa.

```

#include <iostream.h>
class myClass
{
public:
    myClass() // default constructor
    {
        cout << "constructor" << endl;
    }
    ~myClass() // destructor
    {
        cout << "destructor" << endl;
    }
};

int main() {
    myClass array[2]; // static array of objects
    int i = 3;
    myClass *p = new myClass [i]; // dynamic array of objects
    delete [] p;
}

```

```

        return 0;
    }

```

Esse programa gera a saída como mostrado abaixo (2 chamadas ao construtor do array estático e 3 chamadas ao construtor pelo array dinâmico, e depois outras tantas chamadas ao destrutor).

```

constructor
constructor
constructor
constructor
constructor
destructor
destructor
destructor
destructor
destructor

```

9.11.7 Array de objetos e construtor com parâmetros

Uma sintaxe alternativa para se instanciar um objeto chamando construtor com parâmetros é mostrada abaixo. Nesse exemplo, a classe `myClass` possui construtor não padrão, mas não possui construtor padrão. Portanto, a instanciação “natural” de um objeto (instanciação em que não se passa parâmetros para o construtor) não é permitida.

```

class myClass
{
    int m_i;
public:
    myClass(int i) { m_i = i; } // non-default constructor
};

int main()
{
    myClass a; // Error: no appropriate default constructor available
    myClass b = myClass(20); // myClass b(20);
    return 0;
}

```

Usando essa sintaxe, pode-se definir um array estáticos de objetos, e passar parâmetros para cada objeto do array pelo construtor.

```

class myClass
{
    int m_i;
public:
    myClass(int i) { m_i = i; } // non-default constructor defined
};

int main()
{
    myClass a[] = { myClass(1), myClass(3), myClass(10) };
    return 0;
}

```

Não há sintaxe em C++ para um array de objetos alocado dinamicamente (com `new`), em que se possa passar para cada objeto parâmetros pelo construtor.

9.12 Polimorfismo - sobrecarga de operadores

9.12.1 Operador de atribuição (assignment operator – operator=)

Quando se escreve um código com uma operação, o compilador internamente converte esse código pela chamada do método correspondente da classe em questão. Por exemplo: `a=b` (*assignment operator*) internamente é convertido para `a.operator=(b)`; Caso a classe não defina o `operator=`, o compilador define um `operator=` “automaticamente”. Mas se o programador definir o `operator=`, esse método tem preferência sobre o definido pelo compilador.

No exemplo abaixo, o `operator=` é definido automaticamente pelo compilador.

```
#include <iostream.h>
class myClass
{
// empty class
};

int main()
{
    myClass a,b;
    a=b;
    a.operator=(b);
    return 0;
}
```

Esse programa não gera qualquer saída no console.

No exemplo abaixo, o `operator=` é definido pelo programador, e tem precedência sobre o definido pelo compilador.

```
#include <iostream.h>
class myClass
{
public:
    void operator= (const myClass & x)
    {
        cout << "operator=" << endl;
    };
};

int main()
{
    myClass a,b;
    a=b;
    a.operator=(b);
    return 0;
}
```

Esse programa gera a saída como mostrado abaixo.

```
operator=
operator=
```

9.12.2 Porque redefinir o construtor de cópia e operador de atribuição ?

Tanto o construtor de cópia (*copy constructor*) quanto um dos operadores de atribuição (*assignment operator* – `operator=`) são implicitamente criados pelo compilador, mas o programador pode escrever um código seu para esses

métodos. A versão feita pelo compilador funciona bem para muitas classes, copiando o conteúdo de cada atributo. Mas se a classe faz alocação de memória, espera-se tanto do construtor de cópia quanto do operador de atribuição que copie não apenas os atributos, mas também a memória que foi alocada. Nesse caso, a versão dos métodos criada pelo compilador não atende o requisito do programador e torna-se necessário desenvolver explicitamente esses métodos. Veja a seção “Objetos com memória alocada” (9.16.2).

Note que enquanto o construtor de cópia é um construtor, não há retorno desse método. Já o operador de atribuição é um método qualquer, e existe retorno, que eventualmente pode ser void. O retorno do operador de atribuição é algo que ocorre além da atribuição propriamente dita. Uma classe que defina o retorno de operador de atribuição pode escrever um código como mostrado abaixo.

```
#include <iostream.h>
class myClass
{
public:
    myClass operator= (const myClass & x)
    {
        cout << "operator=" << endl;
        return x;
    };
};

int main()
{
    myClass a,b,c,d;
    a=b=c=d; // a.operator=(b.operator=(c.operator=(d)));
    return 0;
}
```

Esse programa gera a saída como mostrado abaixo.

```
operator=
operator=
operator=
```

9.12.3 O único operador ternário

Existem operadores unários (com um parâmetro) e operadores binários (com 2 parâmetros). Em C++ há apenas um operador ternário (com 3 parâmetros), que é o “?” e “:”. O código

```
<1> ? <2> : <3>;
```

é mapeado para o equivalente a

```
if (<1>)
    <2>;
else
    <3>;
```

O programa abaixo é um exemplo de operador ternário “?” e “:”.

```
#include <iostream.h>
int max(int i, int j) {
    return i > j ? i : j;
}
```

```
int main () {
    int k;
    k = max(1,6);
    cout << "k=" << k << endl;
    return 0;
}
```

Esse programa gera a saída como mostrado abaixo.

k=6

O operador ternário não pode ser sobrecarregado.

9.12.4 Operadores binários e unários

Em C++, pelo princípio de polimorfismo, pode-se atribuir um comportamento qualquer a uma operador da lista abaixo. A palavra “sobrecarregar um operador” deve ser entendida como “escrever o método operator.. para a classe em questão”, sendo “..” um dos operadores abaixo.

Operadores de C++ que podem ser sobrecarregados

+	-	*	/	%	^
&		~	!	=	<
>	+=	-=	*=	/=	%=
^=	&=	=	<<	>>	<<=
>>=	==	!=	<=	>=	&&
	++	--	,	->*	->
()	[]	new	delete		

Os operadores que se sobrecarrega em C++ são os operadores unários e binários. Os operadores aritméticos (por exemplo +, -, *, /) podem ser funções globais marcadas pela palavra reservada “friend” (nesse caso há dois parâmetros), ou métodos (com um parâmetro apenas, e atuando sobre o objeto que chama o método). O código gerado por “a+b”, sendo “a” e “b” objetos do tipo myClass é ou o método “a.operator+(b)” ou a função global “operator+(a,b)”. Caso nenhum dos dois seja definido, há erro de compilação. Caso ambos sejam definidos, preferência é dada para o método “a.operator+(b)”. É exatamente isso que ocorre no exemplo abaixo.

```
#include <iostream.h>
class myClass
{
public:

    void operator= (const myClass & x)
    {
        cout << "operator=" << endl;
    };

    myClass operator+ (const myClass & x)
    {
        cout << "operator+ member" << endl;
        return x;
    }

    friend myClass operator+ (const myClass & x, const myClass & y)
    {
        cout << "operator+ global function" << endl;
    }
}
```

```

        return x;
    }

};

int main()
{
    myClass a,b,c;
    a=b; // a.operator=(b);
    a=b+c; // a.operator=(b.operator+(c)); ou operator=(a,operator+(b,c));
    return 0;
}

```

Esse programa gera a saída como mostrado abaixo.

```

operator=
operator+ member
operator=

```

Seja um exemplo de classe relativamente útil abaixo (mas ainda demasiado simplificado), que sobrecarrega operadores aritméticos. Trata-se de uma classe feita para se operar vetores de dimensão 2 (com 2 floats como atributo). O método `operator*` multiplica os vetores com a operação matemática que corresponde a multiplicação de números complexos, isto é, assume que a representação é feita em notação retangular, converte para polar, multiplica os módulos e soma as fases, e em seguida converte para retangular novamente.

```

#include <iostream.h>
#include <math.h> // sqrt
class vetor2
{
public:
    float m_f[2]; // array com 2 float
    vetor2 operator+(vetor2 & obj)
    {
        vetor2 ret;
        ret.m_f[0] = obj.m_f[0] + m_f[0];
        ret.m_f[1] = obj.m_f[1] + m_f[1];
        return ret;
    };
    vetor2 operator*(vetor2 & obj)
    {
        float magThis = sqrt(m_f[0]*m_f[0] + m_f[1]*m_f[1]);
        float magObj = sqrt(obj.m_f[0]*obj.m_f[0] + obj.m_f[1]*obj.m_f[1]);
        float phaseThis = atan(m_f[1]/m_f[0]);
        float phaseObj = atan(obj.m_f[1]/obj.m_f[0]);
        float magRet = magThis * magObj;
        float phaseRet = phaseThis + phaseObj;
        vetor2 ret;
        ret.m_f[0] = magRet * cos(phaseRet);
        ret.m_f[1] = magRet * sin(phaseRet);
        return ret;
    };
}; // end of class vetor2

int main()
{
    vetor2 a, b, ret;
    a.m_f[0] = 1.1;
    a.m_f[1] = 1.2;
    b.m_f[0] = 2.5;

```

```

    b.m_f[1] = 2.6; // carrega dados

    ret = a + b; // ret.operator=(a.operator+(b));
    ret = a * b; // ret.operator=(a.operator*(b));

    cout << "a=" << a.m_f[0] << " ; " << a.m_f[1] << endl; // imprime a
    cout << "b=" << b.m_f[0] << " ; " << b.m_f[1] << endl; // imprime b
    cout << "ret=" << ret.m_f[0] << " ; " << ret.m_f[1] << endl; // imprime ret
    return 0;
}

```

Resultado:

```

a=1.1 ; 1.2
b=2.5 ; 2.6
ret=-0.37 ; 5.86

```

A mesma classe é re-escrita em nova versão abaixo, em que se faz métodos privados para converter de polar para retangular e vice versa. Note o uso do ponteiro this (ponteiro que aponta para o objeto corrente) no uso dessa versão de classe. Veja maiores detalhes sobre o ponteiro this na seção 9.13. Nessa versão, a separação das funcionalidades auxiliares em métodos separados auxilia a manutibilidade da classe.

```

#include <iostream.h>
#include <math.h> // sqrt
class vetor2
{
    vetor2 toRectangular(float mag, float phase)
    {
        vetor2 ret;
        ret.m_f[0] = mag * cos(phase);
        ret.m_f[1] = mag * sin(phase);
        return ret;
    }
    vetor2 toPolar(vetor2 obj)
    {
        vetor2 ret;
        ret.m_f[0] = sqrt(obj.m_f[0]*obj.m_f[0] + obj.m_f[1]*obj.m_f[1]); // magnitude
        ret.m_f[1] = atan(obj.m_f[1]/obj.m_f[0]); // phase
        return ret;
    }
public:
    float m_f[2]; // array com 2 float
    vetor2 operator+(vetor2 & obj)
    {
        vetor2 ret;
        ret.m_f[0] = obj.m_f[0] + m_f[0];
        ret.m_f[1] = obj.m_f[1] + m_f[1];
        return ret;
    };
    vetor2 operator*(vetor2 & obj)
    {
        vetor2 thisPolar = toPolar(*this);
        vetor2 objPolar = toPolar(obj);
        vetor2 ret = toRectangular(thisPolar.m_f[0]*objPolar.m_f[0],
                                   thisPolar.m_f[1] + objPolar.m_f[1]);
        return ret;
    };
}; // end of class vetor2

```

```

int main()
{
    vetor2 a, b, ret;
    a.m_f[0] = 1.1;
    a.m_f[1] = 1.2;
    b.m_f[0] = 2.5;
    b.m_f[1] = 2.6; // carrega dados

    ret = a + b; // ret.operator=(a.operator+(b));
    ret = a * b; // ret.operator=(a.operator*(b));

    cout << "a=" << a.m_f[0] << " ; " << a.m_f[1] << endl; // imprime a
    cout << "b=" << b.m_f[0] << " ; " << b.m_f[1] << endl; // imprime b
    cout << "ret=" << ret.m_f[0] << " ; " << ret.m_f[1] << endl; // imprime ret
    return 0;
}

```

Resultado:

```

a=1.1 ; 1.2
b=2.5 ; 2.6
ret=-0.37 ; 5.86

```

No exemplo abaixo, sobrecarrega-se o operador unário operator! (negação) da classe vetor2, que retorna verdadeiro no caso de os dois floats do atributo m_f, que é um array de 2 floats, serem zero. Caso o operador! não tivesse sido definido, não se poderia compilar a linha “if(!a)”, pois não faz sentido “!a”, sendo “a” um objeto de uma classe definida pelo programador onde o operador! não está definido.

```

#include <iostream.h>
class vetor2
{
public:
    float m_f[2]; // array com 2 float
    vetor2()
    { // default constructor
        m_f[0] = m_f[1] = 0;
    }
    bool operator!()
    {
        return (m_f[0] == 0) &&(m_f[1] == 0);
    }
}; // end of class vetor2

int main()
{
    vetor2 a;

    if (!a) // if (a.operator!())
        cout << "vector not initialized" << endl;
    else
        cout << "vector initialized" << endl;

    a.m_f[0] = 1.1;
    a.m_f[1] = 1.2;

    if (!a) // if (a.operator!())
        cout << "vector not initialized" << endl;
    else
        cout << "vector initialized" << endl;
}

```

```
        return 0;
    }

```

Resultado:

```
vector not initialized
vector initialized

```

9.13 this

A palavra reservada *this* (esse) corresponde a um ponteiro para o objeto em questão. Dentro de uma classe, o seu uso é implícito, e não é necessário usa-lo. Opcionalmente, contudo, pode-se usa-lo. No exemplo abaixo, o construtor foi escrito com duas linhas. Na linha de baixo, repete-se a funcionalidade da linha anterior, apenas usando *this* explicitamente. Desde que *this* é um ponteiro, o acesso a um atributo é feito com o “operador flecha” (->). Lembre-se (*p).a é o mesmo que p->a.

```
class vetor2
{
public:
    float m_f[2];
    vetor2()
    { // default constructor
        m_f[0] = m_f[1] = 0;
        this->m_f[0] = this->m_f[1] = 0;
    }
};

```

Há casos em que é indispensável o uso da palavra reservada *this*. Um desses exemplos é o programa mostrado na página 237.

9.14 lvalue

Um *lvalue* (o termo é a abreviação *left value*, isto é “valor a esquerda”) é uma expressão referindo-se a um objeto.

Numa operação tipo a=b, é necessário que “a” seja um *lvalue*. A generalização desse conceito é a seguinte: É necessário que o identificador a esquerda de um operador de designação seja um *lvalue*. Os operadores de designação (*assignment operator*) são os seguintes:

```
operator=      operator*=      operator/=      operator%=      operator+=
operator-=      operator>>=      operator<<=      operator&=      operator^=      operator|=

```

Alguns operadores retornam *lvalues*. Por exemplo o *operator()*, quando é definido, geralmente retorna um *lvalue*. No exemplo abaixo, uma classe simples de vetor de inteiros com 10 posições é definida. Essa classe *int_vector10* possui o método *operator()* definido, que retorna *lvalue* para *int*, isto é, *int&*. No caso, o retorno de *operator()* é um *int&* que aponta para o array interno da classe, na posição indicada pelo parâmetro de entrada. Um objeto dessa classe pode chamar *operator()* estando tanto a direita quanto a esquerda de um operador de designação (*assignment operator*).

```
#include <iostream.h>

```

```

class int_vector10
{
    int m_a[10];
public:
    int & operator()(int n)
    {
        return m_a[n];
    }
};

int main()
{
    int_vector10 obj;
    obj(3) = 4;
    int k = obj(3);
    cout << k << endl;
    return 0;
}

```

Resultado

4

Seja agora uma variação do programa acima, apenas retirando o “&” do retorno do método `operator()`. Nesse caso, o retorno é um `int`, ou seja, não é um `lvalue`. Portanto, esse operador pode estar do lado direito de um operador de designação, mas não do lado esquerdo.

```
#include <iostream.h>
```

```

class int_vector
{
    int m_a[10];
public:
    int operator()(int n) // essa linha foi modificada em relação ao programa anterior
    {
        return m_a[n];
    }
};

int main()
{
    int_vector obj;
    obj(3) = 4; // erro de compilação !
    int k = obj(3);
    cout << k << endl;
    return 0;
}

```

9.15 Encapsulamento de atributos e métodos

A idéia básica de encapsular dados e métodos é definir cuidadosamente a *interface* de um objeto. Define-se como interface de um objeto o conjunto de atributos e métodos aos quais se tem acesso. Quando se descreve uma classe, pode-se definir 3 níveis de acesso aos atributos e aos métodos:

- **public** – o acesso é permitido para métodos da classe em questão, para classes que herdem da classe em questão e para usuários do objeto.

- `private` – o acesso é permitido apenas para métodos da classe em questão. O usuário de objetos dessa classe não podem acessar um atributo privado diretamente. Classes que herdam propriedades da classe em questão também não podem acessar atributos privados.
- `protected` – o acesso é permitido para métodos da classe em questão, e para classes que herdem dela. Mas o usuário não pode acessar atributos protegidos diretamente.

Para um programador iniciante, pode haver a idéia que tornar todos os atributos como públicos sempre é uma boa técnica de programação. Contudo, a experiência mostra que há necessidade de se proteger o acesso a certos métodos e atributos.

Quando se projeta uma classe ou um conjunto de classes, deve-se pensar em como se pretende que um programador irá usar essas classes. Em princípio, deve-se permitir o acesso somente a atributos e métodos que sejam de interesse, para permitir que a classe cumpra a sua finalidade. Tudo o que não for necessário para o programador deve ficar fora da interface do objeto. Como regra geral, costuma ser boa programação colocar atributos como privados, e oferecer métodos públicos de acesso aos atributos privados, como no exemplo abaixo.

```
#include "vblib.h" // VBString

class person
{
    // private attributes
    VBString m_name;
    VBString m_address;
    VBString m_telephone;
    float m_weight;
public:
    // public access methods
    void setName(VBString name)
    {
        m_name = name;
    };
    void setAddress(VBString address)
    {
        m_address = address;
    };
    void setTelephone(VBString telephone)
    {
        m_telephone = telephone;
    };
    void setWeight(float weight)
    {
        m_weight = weight;
    };

    VBString getName()
    {
        return m_name;
    };
};
```

```

        VBString getAddress()
        {
            return m_address;
        };
        VBString getTelephone()
        {
            return m_telephone;
        };
        float getWeight()
        {
            return m_weight;
        };
    };

int main()
{
    person p1;
    p1.setName("Sergio");
    p1.setAddress("Rua x, número y");
    p1.setTelephone("222-3344");
    p1.setWeight(89.5);

    // ...

    VBString userName = p1.getName();
    VBString userAddress = p1.getAddress();
    VBString userTelephone = p1.getTelephone();
    float userWeight = p1.getWeight();
    return 0;
}

```

O programa acima não faz nada útil, sendo apenas um programa de exemplo. Existe uma classe chamada `person`, que possui 4 atributos privados. Na interface dessa classe, há 8 métodos públicos, para ler e escrever em cada um dos atributos privados da classe.

Qual seria a vantagem de se encapsular (tornar privado) os atributos da classe, já que os métodos tornam-nos públicos de qualquer forma? A vantagem é que tornando os atributos acessíveis apenas através de métodos, com certeza sabe-se que o usuário da classe não irá acessar diretamente os atributos, mas o fará usando os métodos de acesso. Assim, caso surja uma necessidade de upgrade no software (isso sempre ocorre, a experiência o mostra), em que seja necessário, digamos, fazer uma estatística de quantas vezes é acessado o atributo `m_name`. Isso pode ser feito apenas mudando a descrição interna da classe, sem mudar a interface.

No exemplo abaixo, que é um upgrade do exemplo anterior, a classe `person` ganhou um novo atributo chamado `m_getNameTimes`. A classe ganhou também um construtor padrão, chamado automaticamente no momento da criação do objeto, que atribui zero a `m_getNameTimes`. O método `getName` foi alterado internamente, sem que sua funcionalidade fosse alterada, de forma a incrementar `m_getNameTimes` antes de retornar o valor solicitado. Um novo método foi criado para ler o valor de `m_getNameTimes`. Repare que um trecho do programa principal não foi alterado. Esse trecho poderia ser muito grande

num exemplo real de sistema. Caso não existisse a opção de encapsulamento, nada impediria que o usuário da classe person acessasse diretamente o atributo m_name, e com isso seria impossível esse tipo de upgrade.

```
#include "vblib.h"

class person
{
    // private attributes
    int m_getNameTimes;
    VBString m_name;
    VBString m_address;
    VBString m_telephone;
    float m_weight;
public:
    // public access methods
    void setName(VBString name)
    {
        m_name = name;
    };
    void setAddress(VBString address)
    {
        m_address = address;
    };
    void setTelephone(VBString telephone)
    {
        m_telephone = telephone;
    };
    void setWeight(float weight)
    {
        m_weight = weight;
    };

    VBString getName()
    {
        m_getNameTimes++;
        return m_name;
    };
    VBString getAddress()
    {
        return m_address;
    };
    VBString getTelephone()
    {
        return m_telephone;
    };
    float getWeight()
    {
        return m_weight;
    };

    int getNameTimes()
    {
        return m_getNameTimes;
    };

    // default constructor
    person() { m_getNameTimes = 0; }
};

int main()
```

```

{
    person p1;
    p1.setName("Sergio");
    p1.setAddress("Rua x, número y");
    p1.setTelephone("222-3344");
    p1.setWeight(89.5);

    // ...

    VBString userName = p1.getName();
    VBString userAddress = p1.getAddress();
    VBString userTelephone = p1.getTelephone();
    float userWeight = p1.getWeight();

    cout << "No objeto p1, o campo name foi lido " << p1.getNameTimes()
         << " vezes" << endl;
    return 0;
}

```

9.15.1 friend

A palavra reservada “friend” (amigo) é usada para marcar métodos e classes que possam contornar o encapsulamento de dados. No exemplo abaixo, a classe myClass possui um atributo privado. Uma função global fun instancia objetos dessa classe e acessa atributos privados (o que pelo encapsulamento não seria permitido). Mas a função fun é declarada “friend” dentro da classe myClass, e com isso passa a ser permitido que essa função faça acesso a atributos privados dessa classe.

```

class myClass
{
    int m_i;
public:
    friend void fun();
};

void fun() {
    myClass obj;
    obj.m_i = 3; // access to private attribute
}

```

No próximo exemplo, existe uma classe “person”, com atributos privados (como é típico). Uma outra classe “other” possui como atributo um objeto do tipo “person”. Em princípio, nenhum método da classe other pode acessar os atributos privados do objeto da classe person (como é requerido pelo princípio de encapsulamento de atributos e métodos). Portanto o programa abaixo não compila.

```

#include "vblib.h"

class person
{
    // private attributes
    VBString m_name;
    VBString m_address;
    VBString m_telephone;
    float m_weight;
public:
    // public access methods

```

```

void setName(VBString name)
{
    m_name = name;
};
void setAddress(VBString address)
{
    m_address = address;
};
void setTelephone(VBString telephone)
{
    m_telephone = telephone;
};
void setWeight(float weight)
{
    m_weight = weight;
};

VBString getName()
{
    return m_name;
};
VBString getAddress()
{
    return m_address;
};
VBString getTelephone()
{
    return m_telephone;
};
float getWeight()
{
    return m_weight;
};
};

class other
{
    person m_p;
public:
    other()
    { // default constructor
        // direct access to private members of person
        m_p.m_name = "Sergio";
        m_p.m_address = "Rua x, número y";
        m_p.m_telephone = "222-3344";
        m_p.m_weight = 89.5;
    }
};

```

Uma alteração simples no programa acima torna-o compilável. Basta acrescentar a linha de código abaixo dentro da classe person. Essa linha declara que a classe other é “amiga”, isto é, pode acessar as partes privadas da classe em questão (classe person).

```
friend class other;
```

Há outros usos semelhantes para a palavra reservada friend. Um dos usos é marcar métodos e funções globais da classe em questão de forma a permitir que acessem partes privadas da classe. Esse é o caso da sobrecarga de operadores (mostrado em outras seções). Um outro exemplo é mostrado abaixo. Suponha

que para a mesma classe `person`, se deseje escrever uma função global que acrescente um tratamento para o nome da pessoa, isto é, converta “Fulano de Tal” para “Sr. Fulano de Tal”. Isso é facilmente implementável com `VBString`. Caso se deseje acessar diretamente `m_name` o atributo, pode-se escrever a função global como mostrado abaixo.

```
person globalFun(person p) {  
    p.m_name = "Mr. " + p.m_name;  
    return p;  
}
```

Essa função global não poderá ser compilada, pois se está fazendo acesso a atributo privado de objeto da classe `person`. Com a inclusão da linha abaixo dentro da classe `person`, o programa passa a permitir compilação.

```
friend person globalFun(person p);
```

Desde que o uso de `friend` é uma forma de se contornar o encapsulamento, e o encapsulamento é um recurso fundamental da programação OO, existe uma corrente de autores que recomenda evitar o uso de palavra reservada `friend`. Dentro de um certo ponto de vista, pode-se até mesmo estabelecer um critério de qualidade do software desenvolvido em que quanto mais palavras `friend` forem encontradas no código, tanto menos qualidade o software possui. Contudo, o uso de `friend` é um recurso prático e rápido que permite desenvolver software que faz uso de partes privadas das classes com finalidade de, digamos, dar suporte a hardware.

9.16 Alocação de Memória

Existem 3 tipos de variáveis

1. variáveis globais
2. variáveis locais (também chamadas de automáticas)
3. variáveis alocadas dinamicamente

As variáveis **globais** ocupam uma posição arbitrária na memória e podem ter qualquer tamanho, sendo limitadas apenas pela memória do computador. Geralmente não é considerada boa prática de programação a definição de muito espaço em variáveis globais. Isso porque o espaço ocupado pelas variáveis globais é alocado já na carga do programa pelo sistema operacional. Se o espaço de memória global é grande, o programa poderá nem carregar numa máquina com pouca memória.

As variáveis **locais** ocupam uma posição específica da memória - a pilha (*stack*). Quando um programa qualquer é carregado, há um parâmetro que define o espaço de pilha que é reservado para esse programa. A pilha serve também para outras finalidades que guardar variáveis locais. Na pilha guarda-se o endereço das funções de retorno. O tamanho da pilha é fator limitante para saber quantas funções dentro de funções podem ser chamadas, por exemplo. As variáveis locais

são também chamadas de automáticas porque essas variáveis são automaticamente criadas quando se entra numa função e automaticamente destruídas quando se retorna da função. Há um parâmetro do compilador que é o *check stack overflow*. Se um programa é compilado com esse tipo de checagem, toda função primeiro checa o tamanho da pilha e depois usa-a. Se não houver espaço, sinaliza-se erro. Essa checagem toma tempo de processamento. Portanto, para programas com tempo de processamento crítico, uma possibilidade é resetar o flag de check stack overflow. Mas sem essa checagem, caso haja overflow na pilha, o sistema operacional pode congelar.

As variáveis **alocadas** comportam-se como variáveis globais, pois ocupam posição arbitrária na memória. Contudo, há uma diferença importante entre variáveis globais e alocadas. As variáveis globais são definidas em tempo de compilação, enquanto as variáveis alocadas são definidas em tempo de execução. Portanto, o tamanho de uma variável alocada pode ser definido em função de parâmetros que somente são conhecidos em tempo de execução. Isso dá enorme flexibilidade para o programador.

O sistema operacional pode estar executando mais de um programa ao mesmo tempo. A memória, que pode ser alocada dinamicamente, é um dos recursos que o sistema operacional deve gerenciar de forma centralizada para todos os programas que o solicitem. O *heap manager* (*heap* significa monte, pilha, enchimento) é um serviço do sistema operacional que gerencia o espaço livre de memória. Quando o *heap manager* recebe uma solicitação de alocação de algum programa, escreve numa tabela própria a informação de que aquele pedaço de memória está alocado (e portanto não está mais disponível).

A alocação de memória é uma operação que pode falhar, pois pode ser que não haja disponível no momento da alocação um segmento contínuo de memória com tamanho requerido. Portanto um bom programa deve sempre testar se uma tentativa de alocação foi bem sucedida ou não. Um procedimento deve ser previamente definido para ser executado no caso de a alocação de memória fracassar.

9.16.1 Vazamento de memória

Os programas que alocam memória devem ter o cuidado de liberar a memória que não precisam mais. A memória alocada por um programa permanece indisponível para outros usos, mesmo quando o programa que alocou a memória já tenha terminado. Diz-se que um programa que aloca memória e termina sem libera-la possui um “vazamento de memória”. Um conceito análogo existe para um objeto. Se um objeto aloca memória (digamos no construtor) e não a libera (digamos no destrutor), o objeto tem um “vazamento de memória”.

Em C++, a alocação é feita com a palavra reservada `new`, e a liberação é feita com a palavra reservada `delete`. O programa abaixo define uma classe chamada `mydata`,

com alguns dados. A função `main` aloca dinamicamente um objeto da classe `mydata`, usa-o e em seguida libera-o.

```
class mydata {
public:
    double d[333444];
    float f[333];
};

int main () {
    mydata *p_mydata=new mydata; // alloc
    p_mydata->d[222111] = 1.1; // escreve um dado dentro de mydata
    double z = p_mydata->d[222111]; // lê um dado dentro de mydata
    cout << 3 + z << endl; // a resposta será 4.1
    delete p_mydata; // delete
    return 0;
}
```

Além de objetos simples, como acima, pode-se alocar e liberar arrays de objetos. Para isso, basta passar como parâmetro para o operador `new` uma constante ou variável com o número (inteiro positivo) de objetos contidos no array. No exemplo abaixo, uma variável `n` tipo `long` é carregada com a constante 333444 (esse valor poderia ter sido entrado pelo usuário, ou lido de disco). Em seguida, aloca-se um array de `double` de tamanho `n`, usa-se um valor no meio do array para leitura e escrita, e em seguida libera-se a memória.

```
int main () {
    long n=333444;
    double *p_double=new double[n]; // alloc
    p_double[222111] = 1.1; // um valor qualquer no meio
    cout << 3 + p_double[222111] << endl; // 4.1
    delete [] p_double; // delete
    return 0;
}
```

Pode-se verificar se um programa está ou não vazando memória. Para maiores detalhes, veja a seção “verificando vazamento de memória” (página 126). Usando as técnicas descritas nessa seção, remova (ou comente) a linha que deleta a memória nos 2 programas anteriores e verifique que sem essa linha o programa fica vazando memória.

9.16.2 Objetos com memória alocada

Caso se esteja desenvolvendo uma classe que faça alocação de memória, geralmente haverá alocação de memória em algum método (pode ser o construtor), e o destrutor cuida de ver se há memória alocada e se houver, deletar essa memória.

Caso essa classe seja usada em alguma situação de cópia, é preciso que se defina o construtor de cópia e o operador `=` para essa classe. Caso não se defina esses métodos, o compilador automaticamente os substituirá por métodos que copiam os atributos do objeto byte a byte. Mas esse tipo de cópia automática não faz nova alocação de memória para os dados, portanto o objeto copiado não

é uma cópia do original no sentido que geralmente se supõe que deveria ser, para a maioria das aplicações práticas de programação.

Conclusão: ao se desenvolver uma classe que faça alocação de memória, além de definir um método para alocar memória e de deletar essa memória no destrutor, é preciso definir o construtor de cópia e o operador= para que a cópia do objeto seja feita de acordo com o que se espera dele.

Veja o exemplo abaixo. Nesse exemplo, é feita uma classe chamada myVectorAllocMem, que é um vetor de REAL (double), sendo que os dados são alocados (não estão definidos estaticamente na classe). A classe portanto possui um atributo que é ponteiro para os dados, no caso m_data, que é um REAL* (double*). Para simplificar, foi definido um método privado chamado myDelete, que deleta os dados caso o atributo ponteiro para os dados esteja não nulo (isto é, apontando para dados).

Nessa classe há um *default constructor* que também aceita parâmetro implícito. Esse parâmetro é o tamanho do vetor. Se o tamanho é maior que zero, aloca-se dados. Caso contrário, carrega-se zero no ponteiro membro, para garantir que não há nada alocado. O construtor de cópia aloca memória para o novo objeto e copia os atributos (m_size e o conteúdo de memória apontado pelo ponteiro, elemento a elemento).

```
#include <iostream.h>18
#include "vblib.h" // VBMemCheck (Visual C++ only)
#define REAL double
class myVectorAllocMem
{
    REAL *m_data;
    unsigned m_size;
    void myDelete()
    {
        if (m_data)
        {
            delete [] m_data; // delete mem
            m_data = 0;
        }
    }
    void privateConstructor()
    {
        m_data = 0;
        m_size = 0;
    }
}
public:
```

¹⁸ copy data contents = copia conteúdo dos dados, default constructor with default parameter = construtor implícito com parâmetro implícito, the fill method fills the vector with a given value = o método fill (preencher) preenche o vetor com o valor dado, the show method copies the contents of the vector to the console = o método show (mostrar) copia o conteúdo do vetor para o console, a user function that receives a myVectorAllocMem as a parameter, and returns myVectorAllocMem = uma função do usuário que recebe um myVectorAllocMem como parâmetro, e retorna um myVectorAllocMem

```

myVectorAllocMem(int size = 0)
{ // default constructor with default parameter
    privateConstructor();
    m_size = size;
    if (size>0)
        m_data = new REAL[m_size]; // alloc mem
}

myVectorAllocMem(const myVectorAllocMem & obj)
{ // copy constructor
    privateConstructor();
    if (obj.m_size > 0)
    {
        m_size = obj.m_size;
        m_data = new REAL[m_size]; // alloc mem
        for (unsigned i = 0; i < m_size; i++)
            m_data[i] = obj.m_data[i]; // copy data contents
    }
}

virtual ~myVectorAllocMem()
{ // destructor
    myDelete();
}

void operator= (const myVectorAllocMem & obj)
{ // operator=
    if (obj.m_size > 0)
    {
        myDelete();
        m_data = new REAL[obj.m_size]; // alloc mem
        m_size = obj.m_size;
        for (unsigned i = 0; i < m_size; i++)
            m_data[i] = obj.m_data[i]; // copy data contents
    }
    else
    {
        myDelete();
        privateConstructor();
    }
}

// the fill method fills the vector with a given value
void fill(REAL f)
{
    if (m_size > 0)
    {
        for (unsigned i = 0; i < m_size; i++)
            m_data[i] = f;
    }
}

// the show method copies the contents of the vector to the console
void show()
{
    cout << "--- size=" << m_size << endl;
    for (unsigned i = 0; i < m_size; i++)
        cout << m_data[i] << endl;
}

}; // end of myVectorAllocMem

// a user function that receives a myVectorAllocMem as a parameter,

```

```

// and returns myVectorAllocMem
myVectorAllocMem userFun(myVectorAllocMem a)
{
    a.show();
    myVectorAllocMem b(5);
    b.fill(4.4);
    return b;
}

// the main code isolated, to allow memory leak check
void do_all()
{
    myVectorAllocMem z(2);
    z.fill(2.2);
    z.show();
    myVectorAllocMem x;
    x = userFun(z);
    x.show();
}

int main()
{
    VMemCheck w; // mark the memory condition
    do_all(); // do actual processing
    w.check(); // check to see if there is memory leak
    return 0;
}

```

A saída do programa é como mostrado abaixo. Repare que a classe em questão instanciou diversos objetos, que implicitamente chamaram o construtor padrão, construtor de cópia, o operador= e o destrutor. O primeiro vetor mostrado no console é o “z”. O segundo é o “a” (cópia de “z”). O terceiro é “x” (cópia de “b”).

```

--- size=2
2.2
2.2
--- size=2
2.2
2.2
--- size=5
4.4
4.4
4.4
4.4
4.4

```

9.16.3 Array de objetos com construtor não padrão

Um array de objetos pode ser inicializado com um construtor não padrão, desde que seja estático. Não há sintaxe para que se faça alocação de um array de objetos, sendo que cada objeto recebe parâmetros de um construtor não padrão.

No exemplo abaixo, uma classe é definida com um construtor com parâmetro, mas sem construtor padrão. Portanto, não se pode criar um objeto diretamente, como feito com o “obj2” no exemplo abaixo. Mas pode-se criar um obj chamando diretamente o construtor e passando um parâmetro. Pode-se também criar um array estático de objetos, desde que cada objeto seja devidamente inicializado

com a chamada do construtor e tendo um parâmetro para cada construtor de cada objeto.

Mas como não há sintaxe para que se defina a chamada de construtor não padrão para um array de objetos, não se pode alocar myClass.

```
// this class has no default constructor
class myClass {
    int m_i;
public:
    myClass(int i) {
        m_i = i;
    }
};

int main () {
    myClass obj2; // error: no appropriate default constructor available
    myClass obj = myClass(20);
    myClass myArray[3] = { myClass(1), myClass (4), myClass (10) };
    myClass *p;
    p = new myClass [3]; // error: no appropriate default constructor available
    return 0;
}
```

9.17 Criando uma nova classe

Seja o problema de desenvolver uma nova classe, em que se pretende fazer alocação de memória. Considerando a atenção que se deve dar a “construtor padrão”, “construtor de cópia”, “operator=”, “destrutor”, e um local (método) único para se escrever um código comum aos construtores, pode-se escrever um código de esqueleto adequado a uma nova classe que se está criando.

```
class myNewClass
{
    int m_i; // example of private attribute
    void privateConstructor()
    {
        m_i = 0; // initialize attributes (example)
    }
public:
    myNewClass() // default constructor
    {
        privateConstructor();
    }
    myNewClass(const myNewClass & obj) // copy constructor
    {
        privateConstructor();
    }
    virtual ~myNewClass() // destructor
    {
    }
    void operator= (const myNewClass & obj)
    {
    };
};
```

Há que se lembrar que há dois construtores nessa classe. O código comum entre os construtores fica num método privado chamado privateConstructor. O operator= não é um construtor, e não chama privateConstructor. O destrutor

deve ser virtual para o caso de myNewClass ser base de uma classe derivada com destrutor próprio. O fato de o destrutor ser virtual garante (devido a união tardia [*late bind*] que o destrutor da classe derivada seja corretamente chamado quando o objeto derivado sai do escopo.

C++ Multiplataforma e Orientação a Objetos

Parte 4: C++ e Orientação a Objetos (adendo)

10 Biblioteca padrão de C++

10.1 Introdução

A biblioteca padrão de C já existia quando foram desenvolvidos os conceitos de orientação a objetos. Para a linguagem C++ uma nova biblioteca padrão foi criada, onde os conceitos de orientação a objetos são usados. A biblioteca padrão de C++ é mais fácil de ser aprendida e mais segura que a de C. Em princípio, não há motivo para que se use a biblioteca padrão de C nas funcionalidades em que a biblioteca de C++ tem aplicação (a menos que seja para manter código de legado). Nesse capítulo será dada ênfase no uso da biblioteca padrão de C++, e não de C.

Em alguns exemplos, será usada biblioteca não padrão `vblib`. A única diferença técnica entre uma biblioteca não padrão e a biblioteca padrão é o fato de que o programa compilador incorpora automaticamente a biblioteca padrão em qualquer projeto. Uma biblioteca não padrão precisa ser incorporada manualmente. Quase sempre, a finalidade de incorporar essa biblioteca não padrão é usar a classe `VBString`, que é uma classe de string de uso geral. Essa classe é bastante fácil de usar e não requer uso de namespace (como a classe `string` da biblioteca padrão). Para usar a `vblib`, é preciso incluir o header `vblib.h` no fonte onde se deseja usar funcionalidades da `vblib`, e acrescentar o arquivo `vblib.cpp` no projeto em questão.

Tradicionalmente o C manipula strings como array of char. Muitos livros ensinam isso. Mas a experiência mostra que essa forma de programar faz aumentar a probabilidade de bugs grandemente, portanto é geralmente considerada como má programação. A solução recomendada é o uso de uma classe de string na manipulação de strings é boa programação.

Ao usar `VBString` nos exemplos abaixo, além de se exemplificar o que se quer, a mensagem indireta que se está passando é a seguinte:

- Evite o uso de array of char, e manipule strings com uma classe de string como `VBString`.
- Não tenha medo de usar uma biblioteca não padrão, especialmente se o fonte dessa biblioteca estiver disponível.

10.2 Entrada e saída de dados pelo console

Para saída de dados pelo console, usa-se uma corrente (*stream*), que termina em `cout` (console output). A corrente é uma seqüência de elementos separados pelo operador insersor `<<`. Qualquer elemento pode ser inserido na corrente, desde

que para esse elemento esteja definido o operador inseror. Todos os tipos padrão já possuem funções para o operador inseror. Os tipos do usuário também podem entrar na corrente, desde que seja feita a definição de uma função mostrando como o tipo entra na corrente. Esse procedimento é conhecido como sobrecarga do operador inseror e está explicado na seção 10.3.

Veja o exemplo:

Exemplo 1:

```
#include "vblib.h"
int main()
{
    int i = 2;
    float f = 2.2;
    double d = 3.333;
    VBString str = "anything at all";
    cout << "text" << i << " , " << f << " , " << d << " , " << str << endl;
    return 0;
}
```

A saída do programa é mostrada abaixo.

```
text2 , 2.2 , 3.333 , anything at all
```

Analogamente, o extrator >> é o operador inverso ao inseror, e a corrente se inicia em cin (console input). O extrator também pode ser sobrecarregado, da mesma forma que o inseror. Veja o exemplo.

Exemplo:

```
#include <iostream.h>
int main()
{
    int i;
    float f;
    double d;
    cout << "Entre com i:";
    cin >> i;
    cout << endl << "Entre com f:";
    cin >> f;
    cout << endl << "Entre com d:";
    cin >> d;
    cout << endl << "i=" << i << "f=" << f << "d=" << d << endl;
    return 0;
}
```

10.3 Sobrecarga do operador inseror (<<) e extrator (>>)

Quando se cria uma classe, pode-se querer utiliza-la em entradas e saídas do console ou arquivo de forma análoga a tipos padrão. Como o compilador não sabe como usar o inseror e extrator para a classe do usuário, tudo o que se tem a fazer é sobrecarga-los para que entendam a sua classe. Veja o exemplo.

Exemplo:

```
#include <iostream.h>
#include "vblib.h"
class myClass
{

```



```

    int m_i;
    double m_d;
    VBString m_name;
public:
    myClass()
    {
        m_i = 0;
        m_d = 0;
        m_name = "UnNamed";
    };
    friend ostream & operator <<(ostream & stream, const myClass &obj);
    friend istream & operator >>(istream & stream, myClass &obj);
};
// overload insersor
ostream & operator <<(ostream & stream, const myClass & obj)
{
    stream << obj.m_i << endl << obj.m_d << endl << obj.m_name;
    return stream;
}
// overload extrator
istream & operator >>(istream & stream, myClass &obj)
{
    stream >> obj.m_i;
    stream >> obj.m_d;
    stream >> obj.m_name;
    return stream;
}
int main()
{
    myClass a;
    cout << a << endl;
    cout << "Entre i d nome : ";
    cin >> a;
    cout << a << endl;
    return 0;
}

```

Resultado:

```

0
0
UnNamed
Entre i d nome : 1 1.11111 um_nome
1
1.11111
um_nome

```

As funções de sobrecarga do insersor e do extrator *não* podem ser membros da classe respectiva. Isso porque o insersor e o extrator são operatoders (*operator*). Quando um operador é membro de uma classe, o operando esquerdo (implicitamente passado usando o ponteiro *this*) é assumido como o objeto da classe que chama a função *operator*. Lembre-se:

```
a=b; // a.operator=(b) ou operator=(a,b)
```

Contudo, no insersor e no extrator o argumento a esquerda é uma stream, enquanto o argumento a direita é um objeto da classe. O insersor (e extrator) é uma função não membro que precisa ter acesso a elementos internos de uma classe. No caso geral pode haver elementos de acesso privado na classe. A solução para esse dilema é declarar o insersor (e extrator) como uma função *friend* da classe.

10.4 Formatando Entrada / Saída com streams

A forma como C++ gerencia a entrada e saída de dados para console, arquivos, etc. foi toda remodelada em relação a forma como C o fazia. Na biblioteca padrão de C++, existem classes para saída e para entrada, como mostrado abaixo. Além de classes, na biblioteca padrão de C++ há também objetos (ocorrência de classes), como `cout` e `cin`.

Classe de saída da bib. padrão	Objeto	Função
<code>ostream</code> <code>ofstream</code> <code>ostrstream</code>	<code>cout</code> , <code>cerr</code> , <code>clog</code>	Saída em console Saída em arquivo em disco Saída bufferizada em string

Classe de saída da bib. padrão	Objeto	Função
<code>istream</code> <code>ifstream</code> <code>istrstream</code>	<code>cin</code>	Entrada em console Entrada em arquivo em disco Entrada bufferizada em string

O diagrama de heranças entre as classes da biblioteca padrão de C++, no que se refere a entrada e saída formatada, é mostrado na figura abaixo.

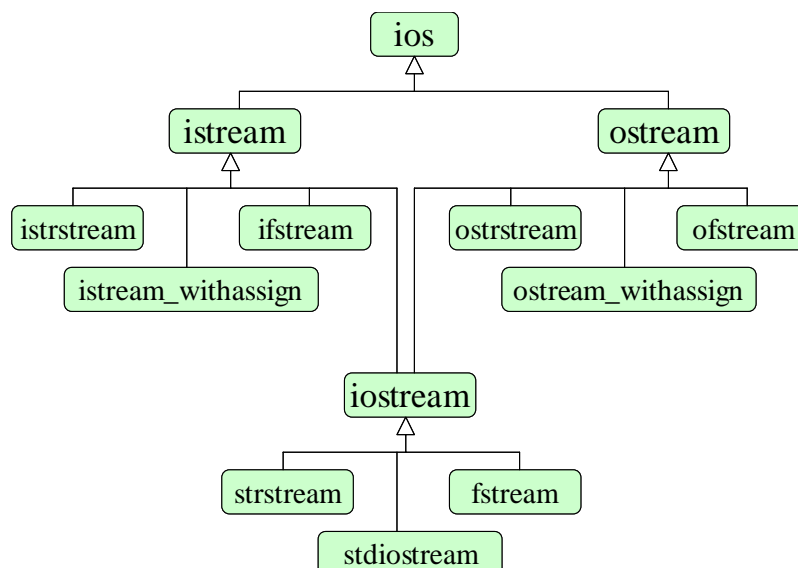


Figura 33: diagrama de heranças entre as classes da biblioteca padrão de C++.

10.4.1 Usando flags de formatação

Formatar a saída significa por exemplo escolher com quantas casas decimais um número vai ser exteriorizado. Em C++ isso é feito usando formatadores e manipuladores de stream, que são usados igualmente para console, arquivo, etc. Todos os streams herdam propriedades da classe ios. Abaixo está mostrado os flags dessa classe que afetam a entrada / saída com streams.

```
// flags de formatação
enum = {
skipws      = 0x0001, // pula espaço em branco na entrada
left        = 0x0002, // saída ajustada a esquerda
right       = 0x0004, // saída ajustada a direita
internal    = 0x0008, // preenche após o sinal ou indicador de base
dec         = 0x0010, // conversão decimal
oct         = 0x0020, // conversão octal
hex         = 0x0040, // conversão hexadecimal
showbase    = 0x0080, // usa o indicador de base na saída
showpoint   = 0x0100, // força o ponto decimal (saída flutuante)
uppercase   = 0x0200, // saída hexadecimal maiúscula
showpos     = 0x0400, // mostra "+" aos inteiros positivos
scientific  = 0x0800, // notação científica
fixed       = 0x1000, // notação ponto flutuante fixa
unibuf      = 0x2000, // descarrega o buffer após a inserção
stdio       = 0x4000, // descarrega stdout e stderr após inserção
boolalpha   = 0x8000, // insere/extrai booleanos como texto ou numérico
};
```

No exemplo abaixo, o flag showpos foi ativado para forçar a exibição do sinal no caso de número positivo.

```
#include <iostream.h>
int main ()
{
    int i = 100;
    cout.setf(ios::showpos);
    cout << i << endl;
    return 0;
}
```

A saída do programa é mostrada abaixo.

+100

No exemplo abaixo, o flag hex foi ativado para forçar a exibição do número em hexadecimal.

```
// com bug
#include <iostream.h>
int main ()
{
    int i = 100;
    cout.setf(ios::hex);
    cout << i << endl;
    return 0;
}
```

A saída do programa é mostrada abaixo.

64

Desde que os flags são definidos em cada bit de uma palavra de 16 bits, mais de um flag pode ser usado ao mesmo tempo, a partir do or bitwise (usando |). Por

exemplo: para usar o uppercase o scientific ao mesmo tempo, pode-se fazer como mostrado abaixo. No mesmo exemplo, mostra-se como desligar um flag usando-se o método unsetf.

```
#include <iostream.h>
int main ()
{
    float f = 100.12;
    cout.setf(ios::uppercase | ios:: scientific);
    cout << f << endl;
    cout.unsetf(ios::uppercase);
    cout << f << endl;
    return 0;
}
```

A saída do programa é mostrada abaixo. Note o ‘E’ (maiúsculo) e o ‘e’ (minúsculo).

```
1.001200E+002
1.001200e+002
```

10.4.2 Examinando os flags de um objeto ios

Para um dado objeto que herda da classe ios, pode-se examinar o conteúdo dos seus flags, a partir do retorno do método “flags”. A partir disso, pode-se criar uma função para exibir a condição de um dado objeto que herda de ios, mesmo que seja o cout da biblioteca padrão. É exatamente isso que faz a função global showIosFlags do exemplo abaixo. Experimente alterar o programa e chamar showIosFlags passando como parâmetro cin.

```
// bug
#include <iostream.h>

void showIosFlags(const ios & iosObj)
{
    cout << "==== ios flag status" << endl;
    long flags = iosObj.flags();
    const char *flagNames[] = {
        "skipws",
        "left",
        "right",
        "internal",
        "dec",
        "oct",
        "hex",
        "showbase",
        "showpoint",
        "uppercase",
        "showpos",
        "scientific",
        "fixed",
        "unibuf",
        "stdio",
        "boolalpha"
    };

    long k = 1;
    for (int i=0 ; i < 16 ; i++) {
        cout << flagNames[i] << " is ";
        if (k & flags)
```

```

        cout << "ON" << endl;
    else
        cout << "off" << endl;
    k <= 1; // rotate k
}
cout << endl;
}

int main ()
{
    showIosFlags(cout);
    cout.setf(ios::uppercase | ios::scientific);
    showIosFlags(cout);
    return 0;
}

```

A saída do programa é mostrada abaixo.

```

==== ios flag status
skipws is off
left is off
right is off
internal is off
dec is off
oct is off
hex is off
showbase is off
showpoint is off
uppercase is off
showpos is off
scientific is off
fixed is off
unibuf is off
stdio is off
boolalpha is off

==== ios flag status
skipws is off
left is off
right is off
internal is off
dec is off
oct is off
hex is off
showbase is off
showpoint is off
uppercase is ON
showpos is off
scientific is ON
fixed is off
unibuf is off
stdio is off
boolalpha is off

```

10.4.3 Definindo o número de algarismos significativos

O método “precision” serve para definir o número de algarismos significativos que se vai usar ao converter um número em ponto flutuante para a stream (corrente). Antes que se defina o número, o objeto cout assume-o como 6.

```

#include <iostream.h>
int main ()

```

```

{
    double d = 1234.123412341234;
    cout << "d=" << d << endl;
    cout.precision(3);
    cout << "d=" << d << endl;
    cout.precision(8);
    cout << "d=" << d << endl;
    return 0;
}

```

A saída do programa é mostrada abaixo.

```

d=1234.12
d=1.23e+003
d=1234.1234

```

10.4.4 Preenchendo os espaços vazios com o método fill

O método “width” (largura) força a largura com que um campo deve ser copiado para a stream. O método “fill” (preencher) define o char que será usado para o preenchimento de espaços vazios (o char padrão é ‘ ’ [espaço]). O exemplo abaixo ilustra o uso desses métodos.

```

#include <iostream.h>
int main ()
{
    double d = 1234.123412341234;
    cout.precision(3);
    cout.width(10);
    cout.fill('*');
    cout << "d=" << d << endl;
    cout.precision(8);
    cout.width(10);
    cout.fill('*');
    cout.setf(ios::left);
    cout << "d=" << d << endl;
    return 0;
}

```

A saída do programa é mostrada abaixo.

```

*****d=1.23e+003
d=*****1234.1234

```

10.4.5 Manipuladores padrão

Além de se chamar funções membro dos objetos que herdam de ios, pode-se também acrescentar à *stream* (corrente) os manipuladores. Na biblioteca padrão de C++ há um conjunto de manipuladores que podem ser usados. Esses manipuladores estão, em geral, declarados no arquivo header padrão iomanip.h.

Manipulador	Entrada/Saída (i/o)	Propósito
dec	i/o	Faz a <i>stream</i> a usar o formato decimal
hex	i/o	Faz a <i>stream</i> a usar o formato hexadecimal
oct	i/o	Faz a <i>stream</i> a usar o formato octal
resetiosflags(long s)	i/o	Desliga os <i>flags</i> especificados por s
setiosflags(long s)	o	

endl		Liga os <i>flags</i> especificados por s
ends	o	Faz a <i>stream</i> inserir um char de fim-de-linha e
flush	o	também descarregar o seu buffer
setbase(int b)	o	Faz a <i>stream</i> inserir um char nulo
setfill(char ch)	o	Faz a <i>stream</i> descarregar o seu buffer
setprecision(int p)	o	Define b como base
	i	Define o char de preenchimento como ch
setw(int w)		Define o número de algarismos significativos como p
ws		Define a largura (<i>width</i>) como w
		Pula espaço em branco a esquerda

O programa da seção 10.4.4 pode ser re-escrito com manipuladores como mostrado abaixo.

```
int main () {
    double d = 1234.123412341234;
    cout << setprecision(3) << setw(10) << setfill('*') << "d=" << d << endl;
    cout << setprecision(8) << setw(10) << setfill('*')
        << setiosflags(ios::left) << "d=" << d << endl;
    return 0;
}
```

A saída é a mesma que a da versão original do programa.

```
*****d=1.23e+003
d=*****1234.1234
```

10.4.6 Manipuladores do usuário

Uma das diferenças no uso de manipuladores e de funções membro de ios é o fato de que é possível criar-se um manipulador do usuário para *streams*. A estrutura para se fazer um manipulador do usuário é muito parecido com a da sobrecarga do operador inserisor (operator<<).

```
// estrutura para se fazer um manipulador do usuário
ostream & myManip(ostream & str [, parameters] ) {
    // user code
    return str;
}
```

No exemplo abaixo, um conjunto de formatações é colocada num manipulador do usuário chamado myStyle.

```
#include <iostream.h>
#include <iomanip.h>
ostream & myStyle(ostream & str)
{
    str.precision(3);
    str.width(10);
    str.fill('*');
}
```

```

        cout.setf(ios::left);
        return str;
    }
    int main ()
    {
        double d = 1234.123412341234;
        cout << myStyle << "d=" << d << endl;
        return 0;
    }

```

A saída do programa é mostrada abaixo.

```
d=*****1.23e+003
```

Vamos agora repedir a funcionalidade do programa da seção 10.4.4, com um novo manipulador do usuário `myStylec` com a passagem de um parâmetro, que é a precisão (número de algarismos significativos). Para isso, é preciso usar-se a macro `OMANIP`, conforme mostrado no exemplo.

```

#include <iostream.h>
#include <iomanip.h>
ostream & myStyle(ostream & str, int precision)
{
    str.precision(precision);
    str.width(10);
    str.fill('*');
    cout.setf(ios::left);
    return str;
}

OMANIP(int) myStyle(int precision)
{
    return OMANIP(int) (myStyle, precision);
}

int main ()
{
    double d = 1234.123412341234;
    cout << myStyle(3) << "d=" << d << endl;
    cout << myStyle(8) << "d=" << d << endl;
    return 0;
}

```

A saída do programa é mostrada abaixo.

```
d=*****1.23e+003
d=*****1234.1234
```

10.4.7 Saída com stream em buffer

Por várias razões, pode ser necessário que um buffer seja criado com o resultado de um *stream* de saída. Isso pode ser feito com a classe `ostrstream`, que está declarada no header padrão `strstream.h` (em Windows), e `strstream.h` (em unix). Há também a classe `istrstream`, para entrada. Admitindo que os programas são compilados em Windows com a diretiva `WIN32` ativada, os programas que usam `strstream` abaixo são multiplataforma.

É preciso tomar cuidado no uso de `ostrstream`, para que se inclua um char nulo de término de string no fim de cada stream, usando-se o manipulador padrão “ends”. Do contrário, a string não estará terminada e o comportamento do

programa poderá ser aleatório. Caso não se inclua o `ends` no final da stream, não haverá nenhum erro ou aviso por parte do compilador. No exemplo abaixo, a mesma funcionalidade da seção 10.4.6 foi repetida, usando-se um objeto da classe `ostrstream`, que por sua vez relaciona-se com um buffer.

```
#include <iostream.h>
#include <iomanip.h>
#ifdef WIN32
    #include <strstream.h>
#else
    #include <strstream.h>
#endif

ostream & myStyle(ostream & str, int precision)
{
    str.precision(precision);
    str.width(10);
    str.fill('*');
    cout.setf(ios::left);
    return str;
}

OMANIP(int) myStyle(int precision)
{
    return OMANIP(int) (myStyle, precision);
}

int main ()
{
    char buffer[500];
    ostrstream myCout(buffer, sizeof(buffer));
    double d = 1234.123412341234;
    myCout << myStyle(3) << "d=" << d << ends;
    cout << buffer << endl;
    myCout << myStyle(8) << "d=" << d << ends;
    cout << buffer << endl;
    return 0;
}
```

A saída do programa é mostrada abaixo.

```
d=*****1.23e+003
d=*****1234.1234
```

A vantagem de se usar o recurso da classe `ostrstream` é o fato de que pode-se manipular strings com todas as facilidades da biblioteca padrão de streams.

A desvantagem de se usar o recurso da classe `ostrstream` é o fato de que é necessário o uso de buffer externo de tamanho fixo. Não é possível que o tamanho do buffer seja alocado automaticamente como no caso de uma classe de string. Caso a stream ultrapasse o tamanho do buffer, o comportamento do programa torna-se aleatório. Isso é portanto uma fonte potencial de bugs.

10.5 Acesso a disco (Arquivos de texto para leitura/escrita)

Há dois tipos básicos de arquivos

1. Arquivos de texto

2. Arquivos binários

A biblioteca padrão de C++, usa as classes `ifstream` e `ofstream` para leitura e escrita em arquivo respectivamente, que são definidas em `fstream.h`. Estas classes possuem funções membro para abrir (`open`) e fechar (`close`) o arquivo em questão e sobrecarrega os operadores `>>` e `<<` respectivamente para a manipulação do arquivo.

Não é preciso, como em C, ajustar uma letra numa string de formato, pois o operador utilizado foi sobrecarregado para todos os tipos padrão do compilador. Para manter a lógica do uso de arquivos com a classe do usuário, basta sobrecarregar o operador `>>` ou `<<` na nova classe.

10.5.1 Escrevendo um arquivo de texto usando a biblioteca padrão de C++

Exemplo:

```
#include <fstream.h>
int main()
{
    float pi = 3.14;
    double d = 2.3456;
    ofstream myfile;
    const char* fname = "fileout.dat";
    myfile.open(fname);
    if (!myfile)
        cout << "File " << fname << " not open";
    myfile << pi << endl << d;
    myfile.close();
    return 0;
}
```

Resultado: cria-se o arquivo `fileout.dat` com o seguinte conteúdo

```
3.14
2.3456
```

10.5.2 Escrevendo um arquivo de texto usando a biblioteca padrão de C

Exemplo:

```
#include <stdio.h>
int main()
{
    float pi = 3.14;
    double d = 2.3456;
    FILE *myfile;
    char* fname = "fileout.dat";
    myfile = fopen(fname, "w");
    if (!myfile)
        printf("File %s not open\n", fname);
    fprintf(myfile, "%f\n%lf", pi, d);
    fclose(myfile);
    return 0;
}
```

Resultado: cria-se o arquivo `fileout.dat` com o seguinte conteúdo

```
3.140000
2.345600
```

Observe que a versão do programa que usa a biblioteca padrão de C++ é substancialmente mais simples que a versão em C. Essa é uma das vantagens do uso do conceito de “orientação a objetos”.

Na versão em C++, usa-se uma corrente (*stream*) direcionada para `cout`. O operador insersor `<<` é definido para todos os tipos padrão (`int`, `float`, `double`, etc) e portanto todos esses tipos podem ser usados na corrente sem necessidade de uma “string de formato” adequada para cada tipo. E mais: qualquer tipo ou classe do usuário pode sobrecarregar o operador insersor de forma que as variáveis do tipo do usuário podem entrar na corrente da mesma forma que os tipos padrão.

Na versão em C, qualquer variável requer o uso de uma string de formato adequada para si. Um erro na definição da string de formato não gera qualquer erro de compilação, mas é gera erros de execução imprevisíveis. No caso do exemplo, o a string de formato contém `%f` associado ao tipo `float` e `%lf` associado ao tipo `double`. Os tipos do usuário devem ser quebrados em elementos que sejam de tipos padrão e então usa-se a função `fprintf` com cada um dos elementos. Isso é muito mais trabalhoso e aumenta muito a possibilidade de erros de programação.

No caso de arquivos de leitura, de forma análoga ao caso de escrita, o uso de C++ e orientação a objetos mostra também a sua vantagem, pois o código é mais fácil de se entender e mais difícil de haver erros.

10.5.3 Lendo um arquivo de texto usando a biblioteca padrão de C++

Exemplo:

```
#include <fstream.h>
int main()
{
    float pi;
    double d;
    ifstream myfile;
    char* fname = "fileout.dat";
    myfile.open(fname);
    if (!myfile)
    {
        cout << "File " << fname << " not open";
        exit(0);
    }
    myfile >> pi >> d;
    cout << "pi=" << pi << endl
        << "d=" << d << endl;
    myfile.close();
    return 0;
}
```

Resultado:

```
pi=3.14
d=2.3456
```

10.5.4 Dica para leitura de arquivo de texto.

Caso se queira ler um arquivo de texto até a sua última linha, é preciso lembrar-se que o flag de “end-of-file” somente é ativado caso se tente ler a linha depois da última. O programa abaixo é um exemplo simples que copia um arquivo de texto para o console.

```
#include <fstream.h>
#include "vblib.h"
int main()
{
    const char* fName = "c:/xxx/t.txt";
    ifstream myFile(fName);
    if (!myFile)
    {
        cout << "Could not open file" << endl;
    }
    VBString a;
    while (true)
    {
        myFile >> a; // read a line to a
        if (myFile.eof())
            break;
        cout << a << endl;
    }
    return 0;
}
```

Admitindo que existe um arquivo chamado “t.txt” no diretório “c:\xxx”, e que esse arquivo contém o que é mostrado abaixo.

Esse é o arquivo t.txt.

Qualquer coisa pode ser escrita aqui.

Mesmo linhas muitíssimo longas podem ser escritas sem nenhum problema

A saída do programa será como mosrado abaixo. Igual ao arquivo de texto t.txt.

Esse é o arquivo t.txt.

Qualquer coisa pode ser escrita aqui.

Mesmo linhas muitíssimo longas podem ser escritas sem nenhum problema

Esse tipo de laço de programa é muito útil para se ler dados de um arquivo de texto, cujo comprimento não se sabe a priori. Por exemplo: leia um vetor de inteiros do disco e coloque o resultado num vetor alocado dinamicamente. Não se sabe antecipadamente a quantidade de dados.

10.5.5 Lendo um arquivo de texto usando a biblioteca padrão de C

Exemplo:

```
#include <stdio.h>
int main()
{
    float pi;
    double d;
    FILE *myfile;
    char* fname = "fileout.dat";
    myfile = fopen(fname, "r");
    if (!myfile)
        printf("File %s not open\n", fname);
    fscanf(myfile, "%f\n%lf", &pi, &d);
}
```

```

        printf("pi=%f\\nd=%lf", pi, d);
        fclose(myfile);
        return 0;
}

```

Resultado:

```

pi=3.140000
d=2.345600

```

10.6 Acesso a disco (Arquivos binários para leitura/escrita)

10.6.1 Escrevendo um arquivo binário usando a biblioteca padrão de C++

```

#include <fstream.h>
int main()
{
    float pi = 3.14;
    // to write, binary mode
    ofstream myfile;
    char* fname = "fileout.dat";
    myfile.open(fname, ios::binary);
    if (!myfile)
        cout << "File " << fname << " not open";
    myfile.write((unsigned char*)&pi, sizeof(float));
    myfile.close();
    return 0;
}

```

10.6.2 Escrevendo um arquivo binário usando a biblioteca padrão de C.

```

#include <stdio.h>
int main()
{
    float pi = 3.14;
    FILE *myfile;
    myfile = fopen("f1.dat", "wb");
    if (!myfile)
        printf("File Not Open\\n");

    fwrite(&pi, sizeof(float), 1, myfile);

    fclose(myfile);
    return 0;
}

```

Prática: Faça um programa que grave um vetor de float com 10.000 posições em 2 arquivos – um arquivo em formato binário e outro arquivo em formato texto.

Resposta:

```

#include <fstream.h>

#define SIZE 10000
#define TYPE float

int main()
{
    TYPE vetor[SIZE];
    int i;
    for (i = 0; i < SIZE; i++)
        vetor[i] = i + 0.1234567890123; // carrega o vetor com qualquer coisa

    // grava o arquivo em binário
}

```

```

ofstream myfile;
char* fname = "f_bin.dat";
myfile.open(fname, ios::binary);
if (!myfile)
    cout << "File " << fname << " not open";

// para gravar um por um dos elementos do vetor
for (int i = 0; i < SIZE; i++)
    myfile.write((unsigned char*)&vetor[i], sizeof(TYPE));

// para gravar o vetor todo de uma vez
// myfile.write((unsigned char*)&vetor[0], SIZE*sizeof(TYPE));

myfile.close();

// grava o arquivo em forma de texto
fname = "f_txt.dat";
myfile.open(fname);
if (!myfile)
    cout << "File " << fname << " not open";

for (i = 0; i < SIZE; i++)
    myfile << vetor[i] << endl;
myfile.close();
return 0;
}

```

Resultado:

Foram criados 2 arquivos. Um é binário, chamado `f_bin.dat`, que não pode ser visto em editor de texto, com tamanho de 40.000 bytes. Outro é chamado `f_txt.dat`, que é em formato texto (portanto pode ser visto em editor de texto), mas seu tamanho é 90.001 bytes.

10.6.3 Lendo um arquivo binário usando a biblioteca padrão de C++

```

#include <fstream.h>
int main()
{
    float pi;
    ifstream myfile;
    char* fname = "fileout.dat";
    myfile.open(fname);
    if (!myfile)
        cout << "File " << fname << " not open";
    myfile.read((unsigned char*)&pi, sizeof(float));
    // usa a variável pi
    myfile.close();
    return 0;
}

```

10.6.4 Lendo um arquivo binário usando a biblioteca padrão de C

```

#include <stdio.h>
int main()
{
    float pi;
    FILE *myfile;
    myfile = fopen("f_bin.dat", "rb");
    if (!myfile)
        printf("File Not Open\n");
    fread(&pi, sizeof(float), 1, myfile);
    // usa a variável pi
    fclose(myfile);
}

```

```

        return 0;
}

```

Prática: Faça um programa para ler os arquivos `f_bin.dat` e `f_bin.dat` feitos na prática anterior.

Resposta:

```

#include <fstream.h>

#define SIZE 10000
#define TYPE float

int main()
{
    TYPE vetor[SIZE];
    int i;
    // lê o arquivo em binário
    ifstream myfile;
    char* fname = "f_bin.dat";
    myfile.open(fname);
    if (!myfile)
        cout << "File " << fname << " not open";

    // para ler o vetor todo de uma vez
    myfile.read((unsigned char*)&vetor, sizeof(vetor));

    // para ler um por um dos elementos do vetor
    // for (i=0 ; i<SIZE ; i++)
    //     myfile.read((unsigned char*)&vetor[i],sizeof(TYPE));

    // usa o vetor para alguma coisa

    myfile.close(); // fecha o arquivo

    // lê o arquivo em forma de texto
    fname = "f_txt.dat";
    myfile.open(fname);
    if (!myfile)
        cout << "File " << fname << " not open";

    // Lê um por um os elementos do vetor
    for (i = 0; i < SIZE; i++)
        myfile >> vetor[i];

    // usa o vetor para alguma coisa
    myfile.close(); // fecha o arquivo
    return 0;
}

```

11 Tratamento de exceção (*exception handling*)

Quando se usa programação orientada a objetos, o programador pode (e deve) usar objetos, que podem ter construtor e destrutor. O tratamento de exceções geralmente é usado no tratamento de algum tipo de erro que foi detectado durante a execução. Possivelmente esse erro conduz a um procedimento de finalização que leva ao fim do programa. Na programação procedural, o tratamento de exceção pode ser feito a partir de um simples goto, mas isso não pode ser feito na programação orientada a objetos, pois pode haver objetos antes do goto que requerem a chamada de destrutor. O goto pura e simplesmente não faz chamar os destrutores dos objetos.

A forma recomendada pela programação orientada a objetos para o tratamento de exceções é o uso de estruturas baseadas nas palavras reservadas abaixo.

- try (tentar)
- catch (capturar)
- throw (jogar)

Durante a execução de um programa, a execução ocorre normalmente de cima para baixo, e pode haver um “bloco try-catch”. Esse bloco é caracterizado por um bloco try seguido de um ou mais blocos catch. Ao fim do último catch após o try, termina-se o “bloco try-catch”. No exemplo abaixo, há bloco try-catch com um bloco try seguido de dois blocos catch. O programa vem sendo executado antes do bloco try-catch, executa o bloco try-catch (de acordo com as regras que discute-se nesse capítulo), e segue sendo executado de cima para baixo.

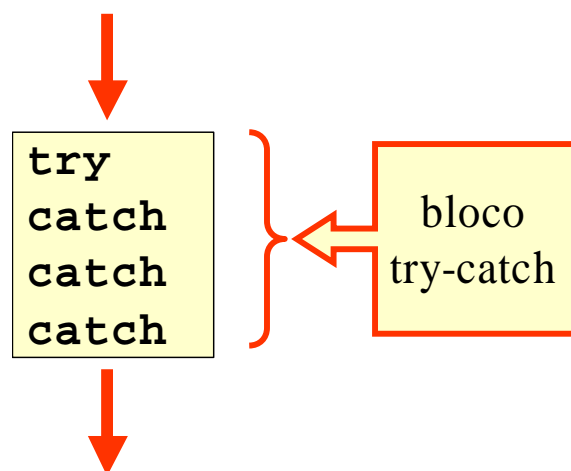


Figura 34: Bloco try-catch


```

#include <iostream.h>

class SomeClass
{
public:
    SomeClass()
    {
        cout << "SomeClass constructor" << endl;
    };
    ~SomeClass()
    {
        cout << "SomeClass destructor" << endl;
    };
};

// function under test
float divide(float a, float b)
{
    SomeClass testObject;
    float ret;

    if (b == 0)
        throw "Can't divide by zero";
    else
        ret = a / b;

    SomeClass testObject2;
    return ret;
}

int main()
{
    cout << "In main." << endl;

    try
    {
        float z = divide(1.1,0);
    }

    catch (char *message)
    {
        cout << "EXCEPTION: " << message << endl;
    }

    cout << "Back in main." << endl;
    return 0;
}

```

Resultado:

```

In main.
SomeClass constructor
SomeClass destructor
EXCEPTION: Can't divide by zero
Back in main.

```

```

// program19
#include <iostream.h>

class CatchHandler
{
public:
    const char *ShowExplanation() const
    {
        return "CatchHandler Explanation";
    }
};

class SomeClass
{
public:
    SomeClass()
    {
        cout << "SomeClass constructor" << endl;
    };
    ~SomeClass()
    {
        cout << "SomeClass destructor" << endl;
    };
};

void FunctionUnderTest()
{
    SomeClass D;
    throw CatchHandler(); // the throw produces an exception
    SomeClass D2;
}

int main()
{
    cout << "In main." << endl;

    try
    {
        cout << "In try block, calling FunctionUnderTest()." << endl;
        FunctionUnderTest();
    }

    catch (CatchHandler E)
    {
        cout << "CatchHandler exception type: "
            << E.ShowExplanation() << endl;
    }

    cout << "Back in main. Execution resumes here." << endl;
    return 0;
}

```

Resultado

```

In main.
In try block, calling FunctionUnderTest ().

```

¹⁹ someProblem = algumProblema, someOtherProblem = algumOutroProblema, code=código, code for someProblem = código para algumProblema, code can throw an exception = o código pode jogar uma exceção

```
SomeClass constructor
SomeClass destructor
CatchHandler exception type: Catch Handler Explanation
Back in main. Execution resumes here.
```

12 RTTI – Identificação em tempo de execução

12.1 Introdução

Um recurso relativamente recente da linguagem C++ é a RTTI (*Runl Time Type Identification*), ou identificação em tempo de execução. Sem esse recurso, a identificação de tipo somente podia ser feita em tempo de compilação. Mas usando RTTI, desde que é possível a identificação de tipo em tempo de execução, novos recursos lógicos podem ser incorporados aos programas.

12.2 Configurando o compilador para RTTI

Alguns compiladores precisam ser configurados explicitamente para que dêem suporte para RTTI. O Visual C++, por exemplo, requer a introdução o comando “/GR” no campo “project options” da caixa de diálogo “Project Settings” (no menu “Project”, de pois “Settings”).

12.3 Palavras reservadas para RTTI

As novas palavras reservadas (ou identificadores da biblioteca padrão) que foram criadas para RTTI são:

- typeid
- dynamic_cast
- static_cast
- const_cast

12.4 typeid

Pode-se saber o nome de um tipo diretamente, ou comparar dois tipos com typeid. No exemplo abaixo, o nome do tipo de 3 variáveis é retornado, e mostrado no console. Tanto tipos padrão quanto tipos do usuário podem ser obtidos. Além disso, os tipos podem ser comparados. No caso, uma função global printSameType recebe um parâmetro booleano e imprime se os tipos são os mesmos ou não no console. O programa de exemplo chama essa função global duas vezes para mostrar o caso em que os tipos em comparação são iguais e o caso em que não são iguais.

```
#include <iostream.h>
#include <typeinfo.h>

class myClass
{
    // ...
```

```

};

void printSameType(bool b)
{
    if (b)
        cout << "Same type" << endl;
    else
        cout << "NOT SAME TYPE" << endl;
}

int main()
{
    int i, j;
    float f;
    myClass a;
    cout << "The type of i is " << typeid(i).name() << endl;
    cout << "The type of f is " << typeid(f).name() << endl;
    cout << "The type of a is " << typeid(a).name() << endl;
    bool sameType = (typeid(i) == typeid(j));
    printSameType(sameType);
    sameType = (typeid(i) == typeid(a));
    printSameType(sameType);
    return 0;
}

```

A saída do programa é como mostrado abaixo.

```

The type of i is int
The type of f is float
The type of a is class myClass
Same type
NOT SAME TYPE

```

A obtenção dos tipos em tempo de execução pode ser de interesse particularmente grande quando se está usando união tardia (*late bind*) entre um ponteiro e um objeto. Com foi explicado na seção 9.10 (página 213).

12.5 typeid e late bind

13 namespace

13.1 Introdução

O conceito de namespace foi introduzido na linguagem C++ para que se evite confusões com nomes de identificadores globais (com visibilidade para todo o programa). Observa-se na prática que há certos identificadores que são “manjados”, isto é, candidatos óbvios a certas aplicações. Identificadores globais tais como `matrix`, `vector`, `list`, `complex`, `string`, `errorHandler`, etc. podem facilmente já ter sido definidos por algum outro desenvolvedor.

No exemplo abaixo, são definidos dois namespaces, sendo que nos dois há um identificador global chamado `errorHandler`, que é uma função. No programa principal, a função `errorHandler` é chamada duas vezes, sendo que o namespace a que pertence a função é explicitado pelo operador de escopo (`::`).

```
#include <iostream>
using namespace std;
namespace oneNamespace
{
    void errorHandler()
    {
        cout << "This is the error handler of oneNamespace" << endl;
    }
}
namespace otherNamespace
{
    void errorHandler()
    {
        cout << "This is the error handler of otherNamespace" << endl;
    }
}
int main()
{
    oneNamespace::errorHandler();
    otherNamespace::errorHandler();
    return 0;
}
```

A saída do programa é como mostrado abaixo.

```
This is the error handler of oneNamespace
This is the error handler of otherNamespace
```

Para simplificar o uso de namespace, existe a palavra reservada “using”. Essa palavra faz referência a um namespace e declara para o compilador que a partir daquele ponto o namespace em questão deve ser tomado implicitamente. Veja o programa acima como ficaria se o namespace “oneNamespace” fosse declarado implícito com “using”.

```
#include <iostream.h>
```

```

namespace oneNamespace
{
    void errorHandler()
    {
        cout << "This is the error handler of oneNamespace" << endl;
    }
}
namespace otherNamespace
{
    void errorHandler()
    {
        cout << "This is the error handler of otherNamespace" << endl;
    }
}
using namespace oneNamespace;
int main()
{
    errorHandler(); // uses namespace oneNamespace by default
    otherNamespace::errorHandler();
    return 0;
}

```

Na “gramática” do C++, nada impede o uso implícito de mais de um namespace, desde que não haja ambiguidade. Por exemplo: o programa abaixo é ambíguo, e não compila;

```

#include <iostream.h>
namespace oneNamespace
{
    void errorHandler()
    {
        cout << "This is the error handler of oneNamespace" << endl;
    }
}
namespace otherNamespace
{
    void errorHandler()
    {
        cout << "This is the error handler of otherNamespace" << endl;
    }
}
using namespace oneNamespace;
using namespace otherNamespace;
int main()
{
    errorHandler(); // error ! ambiguous.
    return 0;
}

```

Caso a função global em questão tivesse nome diferente, então não haveria ambiguidade e nesse caso o programa compilaria normalmente com mais de um namespace implícito.

```

#include <iostream.h>
namespace oneNamespace
{
    void errorHandler()
    {
        cout << "This is the error handler of oneNamespace" << endl;
    }
}
namespace otherNamespace

```

```

{
    void errorHandlerNewName()
    {
        cout << "This is the error handler of otherNamespace" << endl;
    }
}
using namespace oneNamespace;
using namespace otherNamespace;
int main()
{
    errorHandler();
    errorHandlerNewName();
    return 0;
}

```

A saída do programa é como mostrado abaixo.

```

This is the error handler of oneNamespace
This is the error handler of otherNamespace

```

13.2 Namespace aberto

Uma característica interessante do namespace é o fato de que ele é de escopo aberto, ou melhor, expansível. Isso significa que a qualquer momento pode-se expandir o escopo de um namespace, e não é necessário que todos os nomes do escopo de um namespace sejam declarados juntos. No exemplo abaixo, o namespace `oneNamespace` é declarado com uma função global, em seguida o namespace `otherNamespace` é declarado com uma função global de mesmo nome, e posteriormente o namespace `oneNamespace` é expandido com mais uma função global.

```

#include <iostream.h>
namespace oneNamespace
{
    void errorHandler()
    {
        cout << "This is the error handler of oneNamespace" << endl;
    }
}
namespace otherNamespace
{
    void errorHandler()
    {
        cout << "This is the error handler of otherNamespace" << endl;
    }
}
namespace oneNamespace
{ // continuation of oneNamespace
    void otherFun()
    {
        cout << "otherFun of oneNamespace" << endl;
    }
}
int main()
{
    oneNamespace::otherFun();
    return 0;
}

```

A saída do programa é como mostrado abaixo.

otherFun of oneNamespace

13.3 Biblioteca padrão de C++ usando namespace

A própria biblioteca padrão é uma grande usuária de identificadores “manjados”. Portanto, a partir de que o conceito de namespace foi introduzido em C++, os identificadores da biblioteca padrão foram isolados num namespace, chamado de “std” (abreviatura de *standard*, isto é, “padrão”). Muitos headers padrão foram re-escritos para acomodarem-se ao padrão de uso de namespace. Quando um header padrão é re-escrito, o seu nome torna-se o mesmo, apenas eliminando-se a extensão “.h” no final. O programa típico “hello world” (olá mundo), em C++ sem namespace é escrito como abaixo.

```
// hello world in C++ without namespace
#include <iostream.h>
int main ()
{
    cout << "Hello" << endl;
    return 0;
}
```

Esse mesmo programa usando namespace é fica como abaixo.

```
// hello world in C++ WITH namespace
#include <iostream>
using namespace std;
int main ()
{
    cout << "Hello" << endl;
    return 0;
}
```

Outra versão do programa hello world usando namespace é o programa abaixo.

```
// hello world in C++ WITH namespace
#include <iostream>
int main ()
{
    std::cout << "Hello" << endl;
    return 0;
}
```

Para quem quer ou precisa usar namespace a partir da biblioteca padrão, a forma mais fácil e imediata de fazê-lo é substituir a inclusão dos headers padrão com extensão “.h” pelas versões de mesmo nome sem a extensão “.h”, e acrescentar o comando para uso implícito do namespace std, com a linha abaixo (após a inclusão dos headers padrão).

```
using namespace std;
```

O motivo para se usar a versão da biblioteca padrão com namespace é o fato de que algumas funcionalidades mais novas dessa biblioteca somente estão disponíveis na versão com namespace. Por exemplo: a classe de string da biblioteca padrão somente pode ser usada com namespace. Outro exemplo: a classe complex (para números complexos) sofreu algumas melhorias na versão

que é usada com namespace. Desde que se use uma versão moderna do compilador (o que não costuma ser difícil), o uso de namespace não representa qualquer problema.

Há alguns detalhes a serem observados quando se usa namespace std (da biblioteca padrão). O mais importante é que não se deve misturar header sem namespace (com extensão “.h”) com header que tem versão para namespace (sem extensão “.h”). Em outras palavras: caso se esteja usando a biblioteca padrão na versão com namespace, todos os headers padrão devem ser da nova versão. Exemplo:

```
// certo, todos os headers na versão com namespace
#include<iostream>
#include<string>
using namespace std;
// ...
```

Outro exemplo:

```
// errado, misturando versões de header com namespace e sem namespace
#include<iostream>
#include<string.h>
using namespace std;
// ...
```

13.4 Adaptando uma biblioteca existente para uso de namespace std

Seja o caso de um programador que seja autor de uma biblioteca não padrão, foi desenvolvida originalmente sem uso de namespace. Digamos que essa biblioteca se chama mylib, e é composta por dois arquivos: mylib.cpp e mylib.h.

Suponha que essa biblioteca precise da inclusão de algum header padrão, apenas para poder compilar o seu próprio header mylib.h. Isso pode ocorrer caso exista uma classe nessa biblioteca para a qual se sobrecarregou o insersor (operator<<) (veja a seção 10.3, na página 256). No exemplo abaixo, a classe myClass declarou um protótipo da sobrecarga do insersor na sua descrição. Mas para seja possível compilar, o identificador “ostream” precisa estar declarado antes da classe myClass. Como trata-se de um identificador da biblioteca padrão, a solução simples é incluir o header padrão iostream.h. Essa inclusão pode ser feita dentro ou fora do header mylib.h. Vamos supor que seja feita dentro, como mostrado abaixo.

```
// mylib.h
#include <iostream.h>
class myClass
{
    // ...
public:
    friend ostream & operator<<(ostream & s, const myClass & obj);
};
```

Nesse caso, para usar a classe myClass, basta incluir o header da biblioteca, como mostrado abaixo.

```
// main.cpp
#include "mylib.h"
int main ()
{
    myClass a;
    cout << a << endl;
    return 0;
}
```

A questão é: como uma biblioteca como essa deve ser adaptada caso se deseje usar namespace std, sendo que é sabido que não se pode misturar o header padrão com namespace std e sem namespace std ? Uma solução para esse problema é criar uma versão alternativa do header da biblioteca comandada por um define, e compilada de uma forma ou de outra com diretivas de compilação. No exemplo abaixo, a existência da definição de MYLIB_USE_NAMESPACE_STD comanda as duas versões possíveis para o header de mylib.

```
// mylib.h
#ifdef MYLIB_USE_NAMESPACE_STD
    #include <iostream>
    using namespace std;
#else
    #include <iostream.h>
#endif
class myClass
{
    // ...
public:
    friend ostream & operator<<(ostream & s, const myClass & obj);
};
```

Na versão abaixo de utilização da biblioteca mylib, sem uso de namespace std, nenhuma modificação é requerida em relação a primeira versão da biblioteca (desenvolvida sem o conceito de namespace). Se arquivos adicionais da biblioteca padrão forem necessários, basta incluí-los antes da inclusão de mylib.h.

```
// main.cpp, versão sem namespace std
#include <fstream.h> // inclusão de outros headers padrão na versão sem namespace std
#include <string.h>  // inclusão de outros headers padrão na versão sem namespace std
#include "mylib.h"
int main ()
{
    myClass a;
    cout << a << endl;
    // ...
    return 0;
}
```

Usando o mesmo header mylib.h anterior, agora numa utilização que requer o uso de namespace std, basta declarar o define MYLIB_USE_NAMESPACE_STD antes da inclusão de mylib.h. Essa declaração fará com que o header mylib.h use a versão compatível com namespace std. Assim, a compilação é possível, mesmo se for necessário incluir (antes de mylib.h) outros headers padrão na versão com namespace.

```
// main.cpp, versão COM namespace std
```

```

#include <fstream> // inclusão de outros headers padrão na versão COM namespace std
#include <string>   // inclusão de outros headers padrão na versão COM namespace std
#define MYLIB_USE_NAMESPACE_STD
#include "mylib.h"
// daqui para baixo, o programa é igual à versão sem namespace std
int main ()
{
    myClass a;
    cout << a << endl;
    // ...
    return 0;
}

```

Essa técnica é usada para a biblioteca VBLib. O programa abaixo é uma adaptação de um programa anterior, em que a biblioteca VBLib pode ser usada junto com namespace std. Acima da linha “#define VBLIB_USE_NAMESPACE_STD” incluem-se os headers padrão “sem .h”, feitos para serem usados com namespace std.

```

#include <fstream>
#define VBLIB_USE_NAMESPACE_STD
#include "vblib.h"

class myClass
{
    int m_i;
    double m_d;
    VBString m_name;
public:
    myClass()
    {
        m_i = 0;
        m_d = 0;
        m_name = "UnNamed";
    };
    friend ostream & operator <<(ostream & stream, const myClass &obj);
    friend istream & operator >>(istream & stream, myClass &obj);
};
// overload inseror
ostream & operator <<(ostream & stream, const myClass & obj)
{
    stream << obj.m_i << endl << obj.m_d << endl << obj.m_name;
    return stream;
}
// overload extrator
istream & operator >>(istream & stream, myClass &obj)
{
    stream >> obj.m_i;
    stream >> obj.m_d;
    stream >> obj.m_name;
    return stream;
}
int main()
{
    myClass a;
    cout << a << endl;
    cout << "Entre i d nome : ";
    cin >> a;
    cout << a << endl;
    return 0;
}

```

C++ Multiplataforma e Orientação a Objetos

Parte 5: Desenvolvendo para web/cgi usando VBMcgi

14 Programação para web com C++ e VBMcgi

14.1 Introdução

“Web” (teia, no sentido de teia de aranha), é o nome popular da estrutura de programação que usa o protocolo http (*hyper Text Transfer Protocol* - Protocolo de Transferência de Hiper Texto). O software cliente é geralmente chamado de “navegador da teia” (web browser), sendo que os nomes populares são o “Netscape Navigator” e “Internet Explorer”. O computador servidor roda o software “web server”, sendo que os nomes populares são Apache e Microsoft IIS. O http é um protocolo de aplicação criado sobre o protocolo TCP/IP.

VBMcgi é uma biblioteca gratuita para programação CGI para web. As informações e versões atualizadas sobre VBMcgi podem ser obtidas no site <http://www.vbmcgi.org>.

Desde que a web explodiu em importância, em meados da década de 1990, diversas técnicas tem sido disponibilizadas para fazer evoluir os padrões de desenvolvimento. Mas, apesar da quantidade extremamente grande de novidades no mercado de tecnologia de informação, a estrutura original de páginas html se relacionando com programas pela interface CGI ainda é de grande importância.

Resumidamente, CGI (Common Gateway Interface) é a interface que permite páginas web em html executar programas no servidor web, e produzir o retorno no navegador do cliente. Apenas para mencionar um exemplo, os grandes catálogos da Internet tais como Yahoo, Altavista e Cade? são baseados em CGI. Se o internauta digita “banana” no Yahoo para procurar páginas que tenham alguma coisa relacionado a banana, como se poderia supor, há um programa em algum lugar que recebe “banana” como parâmetro. Esse programa é executado no servidor web da yahoo, e a saída é conduzida ao navegador do internauta com o resultado da pesquisa.

Aqueles que projetaram a interface CGI²⁰, de forma bastante inteligente, não associaram a estrutura a nenhuma linguagem de computador em particular. Portanto, qualquer linguagem pode ser usada. Os programas CGI podem ser programas executáveis em binário no servidor, ou arquivos *script*²¹ (arquivos de

²⁰ O protocolo http e a interface CGI foram inicialmente propostos pelo CERN: European Laboratory for Particle Physics. <http://www.cern.ch/>

²¹ *script* significa “roteiro”. Um arquivo *script* é um arquivo de texto, ou seja, não é um arquivo binário.

texto) que são interpretados por um software instalado no servidor. Algumas das opções de linguagem populares para programação CGI com *script* são Perl, PHP, ASP, Cold Fusion. Não há limitação técnica para que se use mais de uma dessas linguagens no mesmo sistema, desde que todos os programas estejam adequadamente instalados no servidor.

Se você usa C++ para programação CGI, a solução óbvia para o início do seu trabalho é escolher uma biblioteca para apoiar o desenvolvimento desse tipo de trabalho. Há muitas bibliotecas C/C++ para programação CGI disponíveis na Internet. Nessa seção, se usará a biblioteca VBMcgi. Trata-se de uma biblioteca gratuita, que disponibiliza todo o código fonte original, multiplataforma (testado em servidores web Unix e Windows). Os programas CGI feitos com VBMcgi podem rodar igualmente em múltiplas plataformas, mas é requerido recompilação.

A biblioteca VBMcgi foi desenvolvida desde o início seguindo uma filosofia principal:

“regra de ouro” da VBMcgi:
isolar o trabalho do webmaster e do webdesigner

Essa característica não é encontrada em outras bibliotecas C++ para programação CGI. Por causa dessa regra de ouro, qualquer software de webdesign comercial (que salva código html) pode ser usado. Mais importante: o webmaster é o único que precisa saber de C++, VBMcgi e assuntos relacionados. Por causa dessa regra de ouro, o webdesigner não precisa saber de nada disso. Por seguir algumas instruções bastante simples, o webdesigner pode fazer seu trabalho num time junto com o webmaster. Mesmo que o webmaster e o webdesigner sejam a mesma pessoa, a regra de ouro ainda ajuda muito no desenvolvimento de um sistema de forma a ser de fácil manutenção. A regra de ouro é implementada a partir de duas funcionalidades (*features*) principais, como será explicado em detalhes mais tarde.

- “string change feature” (veja seção 14.6.5)
- “call function feature” (veja seção 14.6.6)

O nome VBMcgi significa “Villas-Boas and Martins library for CGI” - os dois autores originais da biblioteca. Lembre-se VB significa “Villas-Boas”, e não “Visual Basic”.

Para testar os programas CGI, é necessário que o leitor tenha condições de colocar os executáveis num diretório mapeado por um programa web server, e usar os programas clicando num navegador. Para quem usa Windows, pode-se obter uma cópia gratuita do apache para Windows, ou usar o IIS da Microsoft, e

trabalhar num computador isolado, fazendo a “Internet de um computador só”. Isto é, aponta-se o navegador para “localhost” e com isso literalmente navega-se na teia (*browse the web*) no próprio computador, sem necessariamente precisar estar conectado na Internet para o trabalho de desenvolvimento.

Para facilitar para o leitor iniciante, há uma seção inicial introdutória sobre web.

14.2 Navegando na teia

Vamos fazer uma rápida revisão do que realmente acontece quando alguém navega na teia. Suponha o internauta quer ver uma página no domínio “name.com”. Ele digita “http://name.com” no campo URL (universal resource locator) do navegador (cliente de http), e vê o resultado.

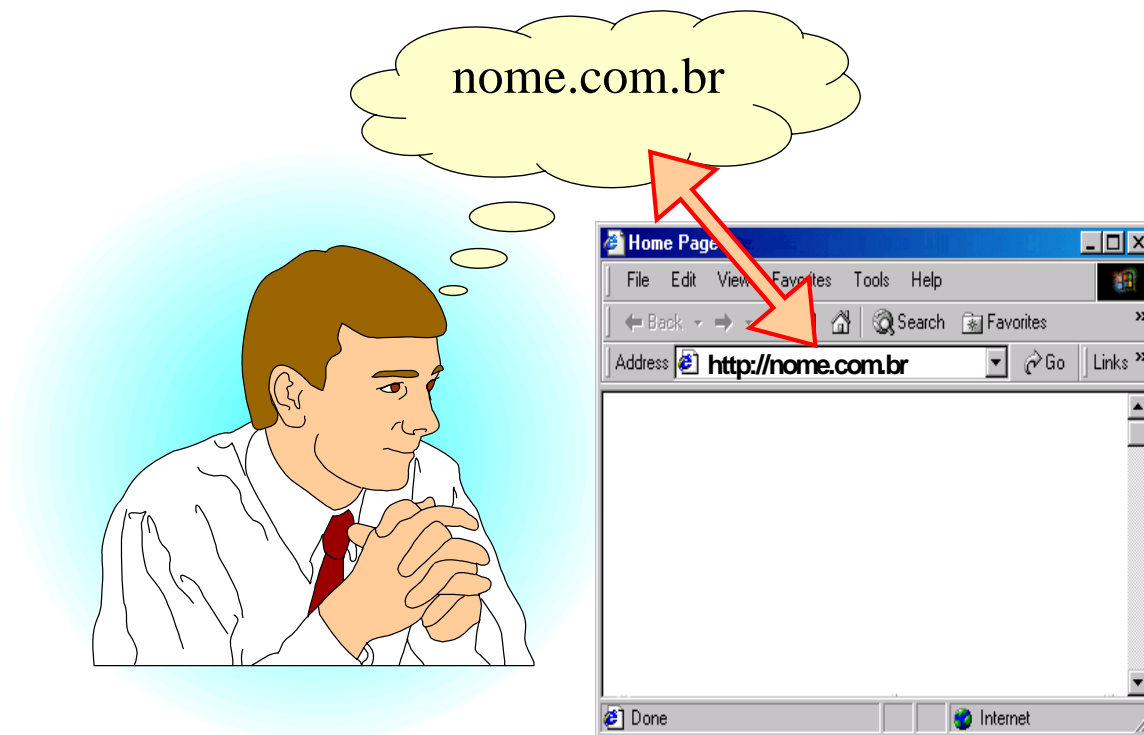


Figura 35: O internauta digita a URL no navegador e vê o resultado

A partir do que vê, o internauta clica em ligações (*links*) e com isso faz novas requisições ao servidor web. Suponha que o cliente clique em algo relacionado ao arquivo “arquivo.html”. Isso fará enviar uma mensagem de “mande-me o arquivo ‘arquivo.html’ ” para o servidor, que responderá enviando o arquivo “arquivo.html” para o cliente.

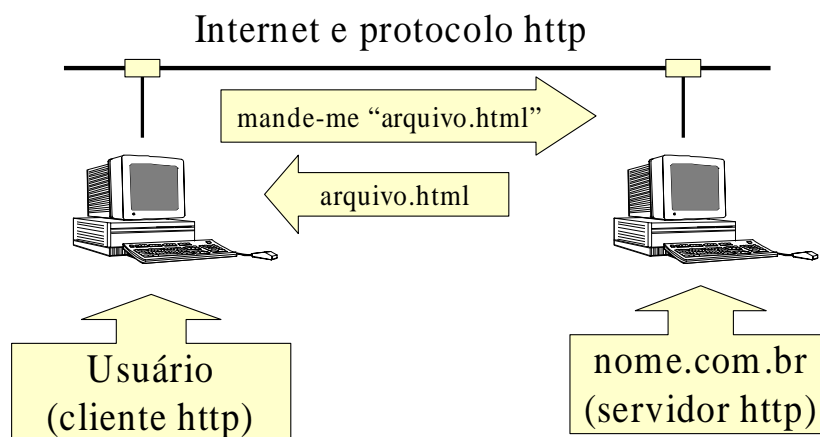


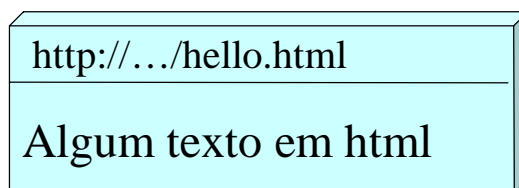
Figura 36: O cliente faz requisição de dados e o servidor os envia.

É importante lembrar que a navegação na teia ocorre SEM que se especifique qual é o software do navegador nem qual é o software servidor, ou mesmo SEM que se especifique qual é o sistema operacional de qualquer dos computadores (cliente e servidor). A única padronização é o protocolo TCP/IP e http, com todos os seus detalhes. Também é padronizada a linguagem HTML, para conteúdo de páginas web²².

Para criar uma página web é simples. Basta escrever os dados na linguagem html e salvar num arquivo, geralmente com extensão *.html ou *.htm. O arquivo "hello.html" abaixo mostra uma página simples no formato html.

```
<!-- hello.html -->
<html><body>
Algum texto em html
</body></html>
```

O resultado é mostrado no navegador como abaixo²³.



Para criar uma ligação (*link*) é muito simples. Basta acrescentar um tag na linguagem html. No exemplo abaixo, os arquivos a.html e b.html estão apontando um para o outro.

```
<!-- a.html -->
```

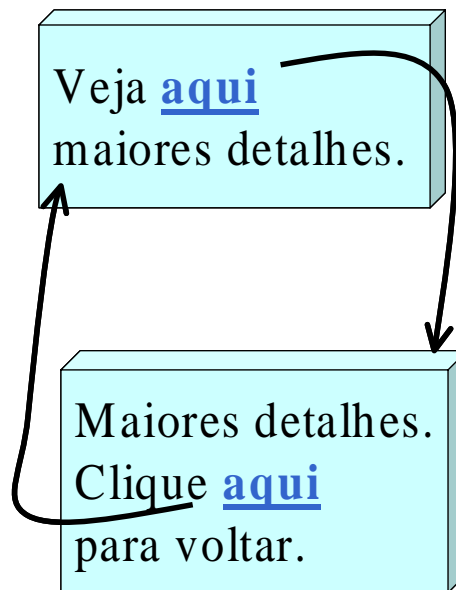
²² Nos dias atuais, algumas padronizações adicionais são requeridas pela web. Mas para manter o estudo simples por enquanto, apenas essas padronizações são apresentadas.

²³ Essa forma "de caixa" é representação do navegador.

```
<html><body>
Clique <a href="b.html">aqui</a> para detalhes.
</body></html>
```

```
<!-- b.html -->
<html><body>
Mais detalhes. <br>
```

```
Clique <a href="a.html">aqui</a> para voltar.
</body></html>
```



14.2.1 CGI

A interface CGI – Common Gateway Interface – permite que uma página html chame um programa (pode ser um executável binário ou um script) no servidor, e enviar o resultado para o navegador do cliente. A interface CGI não impõe o uso de nenhuma linguagem de computador em particular, sendo que as mais usadas são C/C++, Delphi, Perl, ASP, PHP, JSP (Java), Cold Fusion. Os exemplos mostrados nesse texto são feitos com a linguagem C++ e a biblioteca VBMcgi.

Considere o código simples abaixo. Usando um compilador, o arquivo fonte hello_cgi.cpp é construído (*build*) para o arquivo executável hello_cgi.cgi²⁴. Os

²⁴ Em unix, qualquer extensão de arquivo pode ser executável. Em Windows, a maioria dos compiladores produz implicitamente extensão *.exe para executáveis. Mas esses arquivos podem ser renomeados para *.cgi, e o web server em Windows faz esses arquivos serem executados de forma equivalente a um arquivo *.exe.

programas CGI em geral são arquivos executáveis com extensão *.cgi, que são arquivos executáveis em linha de comando sem nenhuma característica especial.

```
// hello_cgi.cpp
#include <iostream.h>
int main () {
    cout << "Content-type:text/html" << endl << endl;
    cout << "<html><body>" << endl;
    cout << "Some text in html" << endl;
    cout << "</body></html>" << endl;
    return 0;
}
```

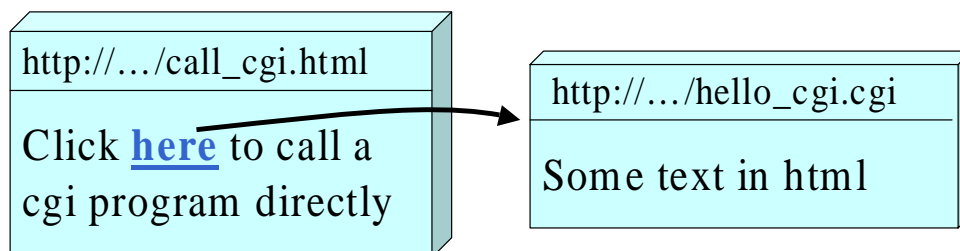
O programa hello_cgi.cgi, se executado em linha de comando, dá o resultado abaixo.

```
Content-type:text/html
```

```
<html><body>
Some text in html
</body></html>
```

O código HTML abaixo chama o cgi como se fosse um arquivo html normal.

```
<!-- call_cgi.html -->
<html><body>
Click <a href="hello_cgi.cgi">here</a> to call a cgi program directly
</body></html>
```



14.3 Escolha de uma tecnologia para web/cgi

Nessa seção se discute as principais tecnologias que existem para web/cgi, com vantagens e desvantagens. Não creio que haja uma resposta definitiva para a pergunta de “qual é a melhor tecnologia”. Existem muitos casos, e para cada um deles a melhor escolha pode ser diferente.

- ASP - é uma tecnologia de cgi baseado em script, com marca Microsoft. A vantagem é que essa tecnologia vem integrada a ambiente de desenvolvimento da própria Microsoft. É bastante fácil de aprender a usar. Há muitos profissionais no mercado que conhecem essa tecnologia, o que deixa o gerente de tecnologia em situação de baixo risco nesse quesito. O uso de ASP na prática implica em usar servidores Windows, portanto essa opção amarra a base tecnológica na necessidade de se comprar licenças de Windows. É boa opção para quem decidiu estrategicamente adotar

soluções com marca Microsoft, e má opção caso contrário. A linguagem ASP somente serve para programação web/cgi.

- Perl - trata-se de uma linguagem script que existia antes de se inventar a web e cgi. A partir do surgimento da web, perl foi adaptada para cgi, e conta atualmente com versão para unix e Windows. É menos desenvolvida que PHP para aplicações cgi, mas conta com tradição mais antiga. Pode ser boa opção para quem possui cultura histórica de uso de perl.
- PHP - é uma tecnologia desenvolvida especificamente para cgi baseado em script, que é gratuita e com versão para Windows e Unix. Há inúmeras referências para essa tecnologia, tanto na web quanto em literatura técnica. Há abundante oferta de componentes para PHP. A linguagem PHP somente serve para web/cgi. É uma boa opção para quem quer fazer cgi com script, e se interessa pela possibilidade de usar servidores Windows ou unix.
- Cold Fusion - É um produto pago para cgi com script, originalmente da empresa allaire, depois encampada pela empresa Macromedia. Consta que é muito fácil de aprender a usar. Há opções gratuitas muito boas, portanto a opção por essa tecnologia possivelmente faz sentido a partir de um motivo adicional, por exemplo existência de uma cultura no uso desse produto.
- JSP e Java Beans - É uma tecnologia para cgi com script, que pode ser conectada a componentes escritos em Java (os tais *Java Beans* - “grãos de café”). Como os componentes são usado em formato binário intermediário (*bytecode*), a característica multiplataforma de Java (escreva uma vez e execute em qualquer lugar) vale para esses componentes. Uma característica importante dessa opção é a existência de rede de alianças vigorosa em torno desse padrão. Nessa rede entram grandes empresas, incluindo Sun, IBM, Oracle, RedHat, etc. Uma ausência importante nessa aliança é a Microsoft. A partir dessa rede de alianças em favor de JSP, muitas empresas grandes adotaram essa tecnologia como padrão para novos projetos. O uso de Java Beans dá grande flexibilidade ao desenvolvimento. A partir do uso dessa tecnologia, surge oportunidade de comercialização de componentes para web (os *beans*). Uma desvantagem dessa tecnologia é que ela é menos fácil de se aprender que alternativas como ASP ou PHP. O resultado disso é a escassez de profissionais qualificados para a atividade, e conseqüentemente o potencial aumento do custo de projetos baseados nessa tecnologia. Mas pode ser uma boa opção para quem tem como decisão tecnológica conhecer bem Java.
- C++ e VBMcgi. É opção para quem tem como decisão estratégica conhecer bem e trabalhar com C/C++. Quem tem conhecimento em C/C++ ou quer incrementa-lo pode desenvolver software para web/cgi

usando essa linguagem. É também boa opção quando se quer desenvolver um sistema baseado em web/cgi que faça interface com hardware do servidor, que geralmente é feita em C/C++. Por se usar desde o primeiro momento a própria linguagem C/C++, o acesso direto ao hardware por programas cgi é imediato. A biblioteca VBMcgi isola o webmaster e o webdesigner. Com isso, um programador C++ pode integrar um time de desenvolvimento de sistema para web com parceiros - particularmente o webdesigner - que não conheçam C++. É uma tecnologia disponibilizada a partir do ano 2000, e ainda com relativamente poucos usuários. Mas o número de usuários tem crescido rapidamente, e a partir da disponibilização desse livro deve crescer muito mais ainda.

14.4 História e futuro da VBMcgi

A biblioteca VBMcgi é um projeto idealizado e mantido pelo autor do livro. Essa biblioteca foi feita do zero (*from scratch*), sendo que a versão 1.0 foi o trabalho de projeto final de Alessandro Martins, feito sob minha orientação. Esse trabalho foi disponibilizado em Novembro de 2000. Pouco depois registrou-se o domínio vbmcgi.org, e fez-se uma página para distribuição da biblioteca VBMcgi. Essa página contém sempre a última versão da biblioteca, dos tutoriais, etc.

Desde a disponibilização da primeira versão, a VBMcgi tem sido objeto de ensino para os alunos do DEL - Departamento de Engenharia Eletrônica e Computação da UFRJ [65]. Os estudantes testam-na e dão sugestões preciosas de melhorias. Diversos projetos acadêmicos e comerciais tem usado a biblioteca VBMcgi com sucesso. Aos poucos, usuários diversos - não só no Brasil - tem usado a VBMcgi e sugerido melhorias.

Em Março de 2001, foi disponibilizada a versão 1.3. Foram feitas modificações na VBMcgi e na VBLib para que ambas se tornassem compatíveis com STL (C++ Standard Template Library) e com namespace.

Em Maio de 2001, foi disponibilizada a versão 1.6. Nessa versão acrescentou-se a classe VBCalendar na VBLib, e a classe VBtableMonth na VBMcgi. Foram também acrescentados métodos na VBMcgi para checagem de cartão de crédito, CIC, email, telefone e outros.

Em 14 de Janeiro de 2002, foi disponibilizada a versão 2.0. Foram concertados os bugs: uma variável global (estática) previne a possibilidade de múltiplos headers ocorrerem na saída. Mudanças: agora o método formDecode usa o método scapeSequenceReverse de VBString. A classe VBMcgi agora possui construtor de cópia, e por isso o objeto pode ser passado como parâmetro por valor se for necessário (e não apenas como referência). Novas funcionalidades: novas classes VBPageCount e VBMenuHtml, novo método "out", que é abreviação de "changeAndStdOutCall". A lista de pares de strings do formDecode e do string change feature agora são dois objetos independentes. Agora todos os arquivos

fonte da biblioteca iniciam-se por VBMcgi (exceto os arquivos da VBLib, que começam por vblib.*).

Em 14 de Janeiro de 2002, foi disponibilizada a versão 2.1. Na nova distribuição agora inclui-se projeto para Visual C++, Borland Builder C++ e gnu C++.

Em 28 de Janeiro de 2002, foi disponibilizada a versão 2.2. Resolveu-se um bug no controle do html header no caso de saída não para console, mas para arquivo. O método formDecode agora implicitamente assume o valor false.

(Em breve futuro) - Será disponibilizado uma biblioteca middleware para acessar a maioria dos produtos de banco de dados (incluindo Oracle, MS SQL server, DB2, Informix, ODBC, MySql, PostgreSQL). Além disso, novos métodos para upload e download, e controle de seção.

14.5 Instalando VBMcgi

Nessa seção, é mostrado como instalar a biblioteca VBMcgi em alguns casos. Trata-se de uma biblioteca que pode virtualmente ser usada em qualquer computador. Nessa versão de tutorial, serão apresentados apenas 3 casos, sempre ligando estaticamente a biblioteca VBMcgi com o programa executável - Em unix com o compilador gnu, e em Windows com o compilador Visual C++ 6.0 sp5, e Borland Builder 5.0.

14.5.1 Windows, Visual C++

Esse tutorial foi feito para Visual C++ versão 6.0, com service pack 5. É sabido que o Visual C++ 6.0 sem o uso de service pack não compila a VBMcgi.

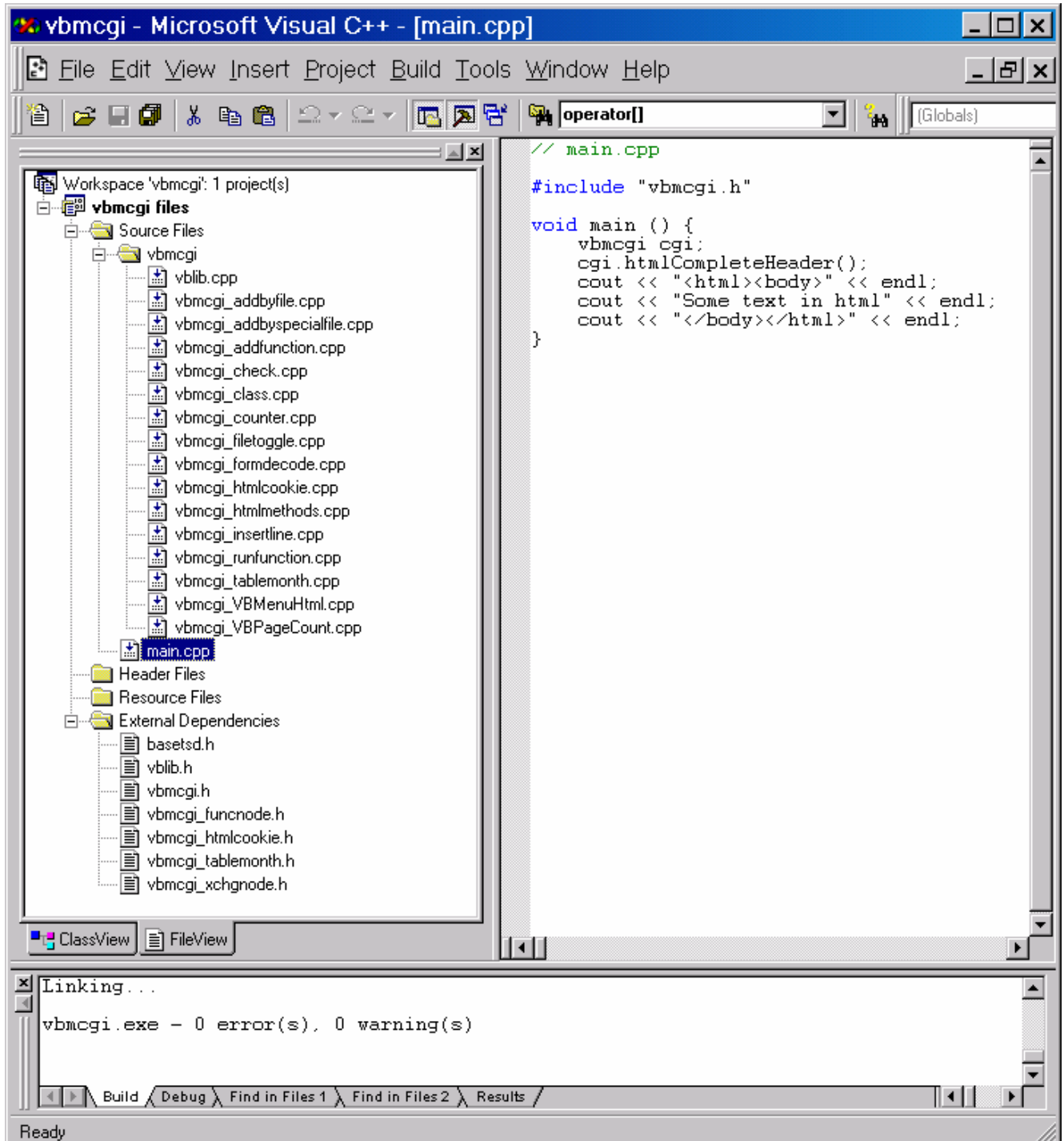
Na distribuição do arquivo vbmcgi22.zip há um projeto pronto para Visual C++ 6.0 sp5. Descomprima (unzip) todos os arquivos num diretório de trabalho e abra o arquivo vbmcgi.dsw (um workspace com um projeto para vbmcgi). Esse projeto deverá compilar e executar. Edite o arquivo main.cpp livremente, para rodar os exemplos.

Se quiser abrir um novo projeto para VBMcgi, selecione no menu File - New, escolha o tab "Projects". Escolha "win32 console application".

O arquivo main.cpp contém um exemplo de hello para VBMcgi.

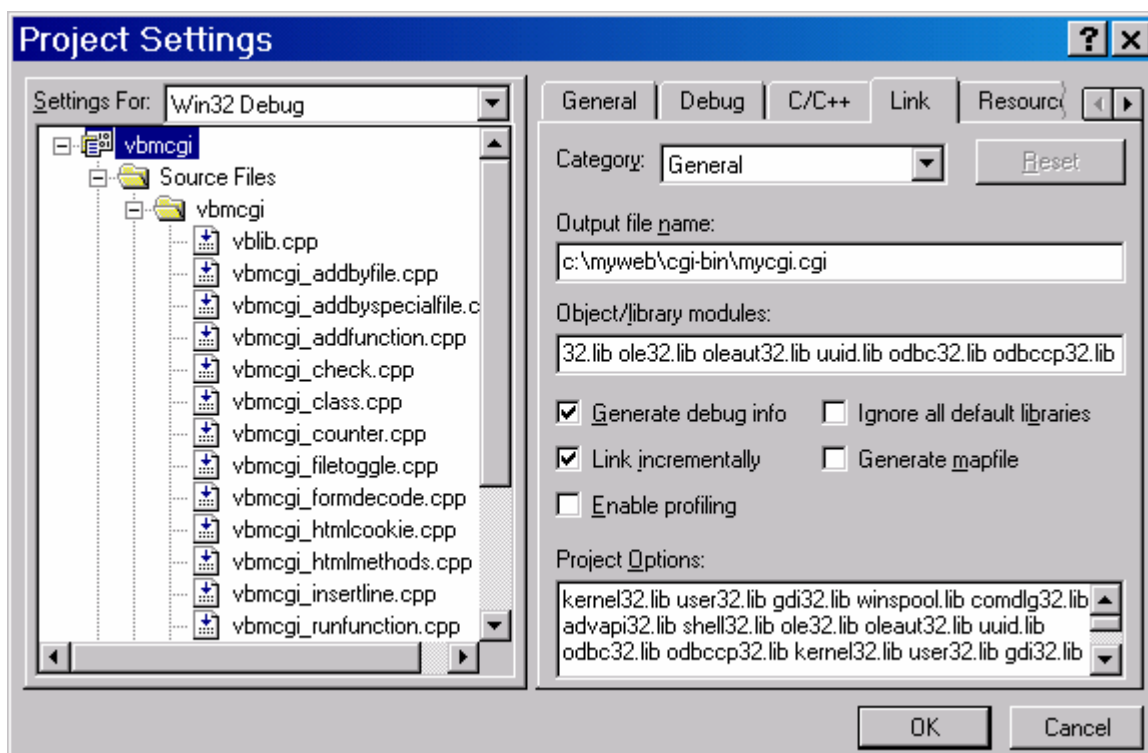
```
// main.cpp
#include "VBMcgi.h"
int main () {
    VBMcgi cgi;
    cgi.htmlCompleteHeader();
    cout << "<html><body>" << endl;
    cout << "Some text in html" << endl;
    cout << "</body></html>" << endl;
    return 0;
}
```

Para construir e executar o projeto, pressione F7. Execute com <ctrl-F5>. O workspace deve ter um aspecto como na figura abaixo (em file view).

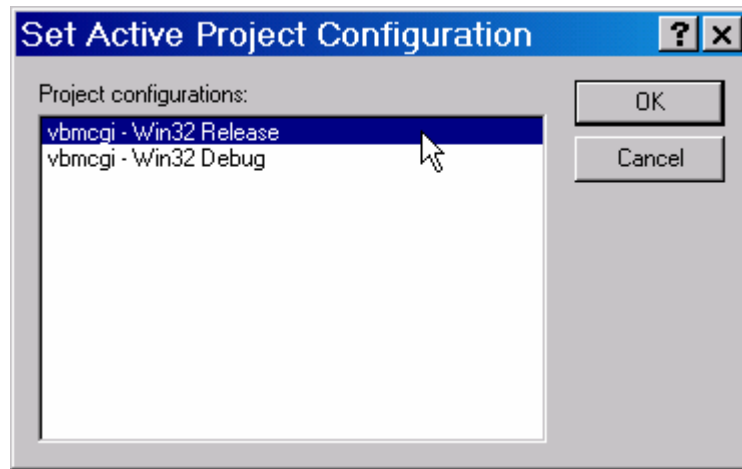


Supõe-se que há um servidor web instalado no seu computador, por exemplo Apache. Há versão gratuita desse servidor web para Windows, disponível em

www.apache.org. Na configuração do servidor web, há um diretório (ou mais) para execução de programas cgi. Suponha que esse diretório para programas cgi seja “c:\myweb\cgi-bin”. Pode-se fazer o Visual C++ gerar um programa cgi diretamente no diretório que se quer por selecionar no menu Project – Settings (ou <alt-F7>), no tab “link”, category “General”, preencha o campo “output file name”. Por exemplo, preencha esse campo com “c:\myweb\cgi-bin\mycgi.cgi”. Agora, após “build” (F7), o programa cgi será atualizado.



Uma dica útil é selecionar a versão “release” para o programa cgi, ao invés da versão “debug”. No desenvolvimento de programas cgi, a maior parte do processo de debug ocorre por executar o programa cgi com o web server. Portanto, as funcionalidades internas do Visual C++ para debug não são usadas. Por selecionar “release” como versão de saída, o programa cgi fica muito menor que no caso de versão “debug”. Para escolher a versão “release” para saída, selecione no menu Build – Set Active Configuration, e selecione “release” na caixa de diálogo, como mostrado na figura abaixo.



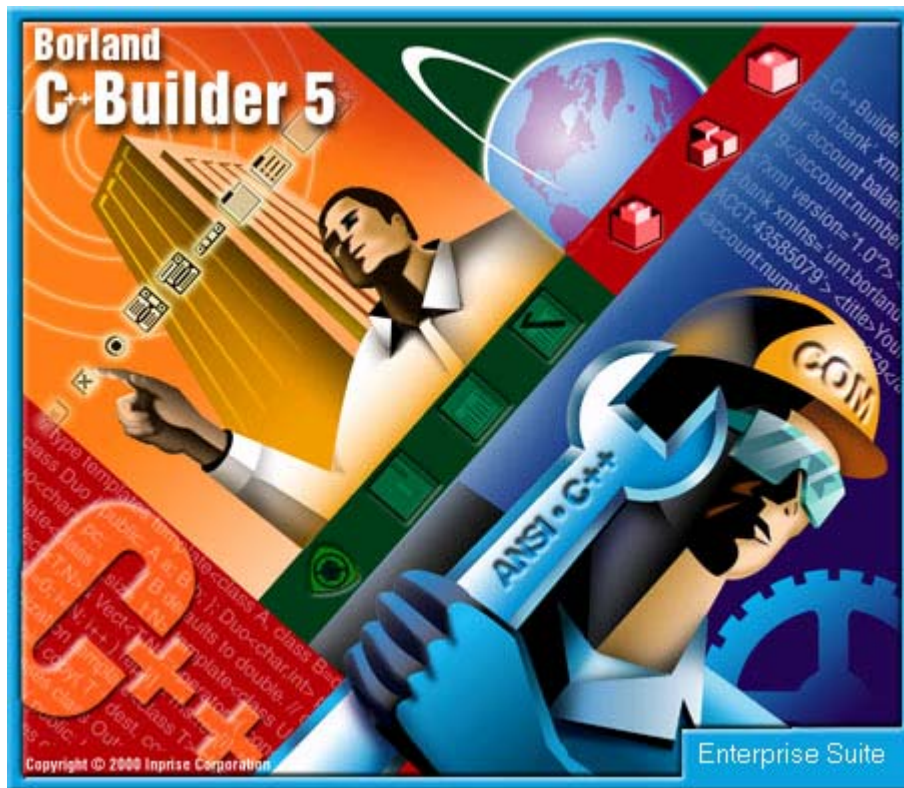
Suponha que se está desenvolvendo um software para web para ser hospedados num provedor comercial de presença. Desde que seu compilador é o Visual C++, certifique-se que o sistema operacional do servidor do seu provedor de presença é o Windows. Desenvolva e teste localmente o seu sistema usando apenas um único computador com sistema operacional Windows. Quando o resultado estiver satisfatório, faça upload dos arquivos html e cgi para o provedor, e com isso coloque o sistema no ar. Não há necessidade de fazer o upload do código fonte dos programas cgi, mas apenas os executáveis (*.cgi ou *.exe).

Por rodar o programa de exemplo dentro do Visual C++ 6.0 sp5, a saída deve ser aproximadamente como mostrado abaixo.

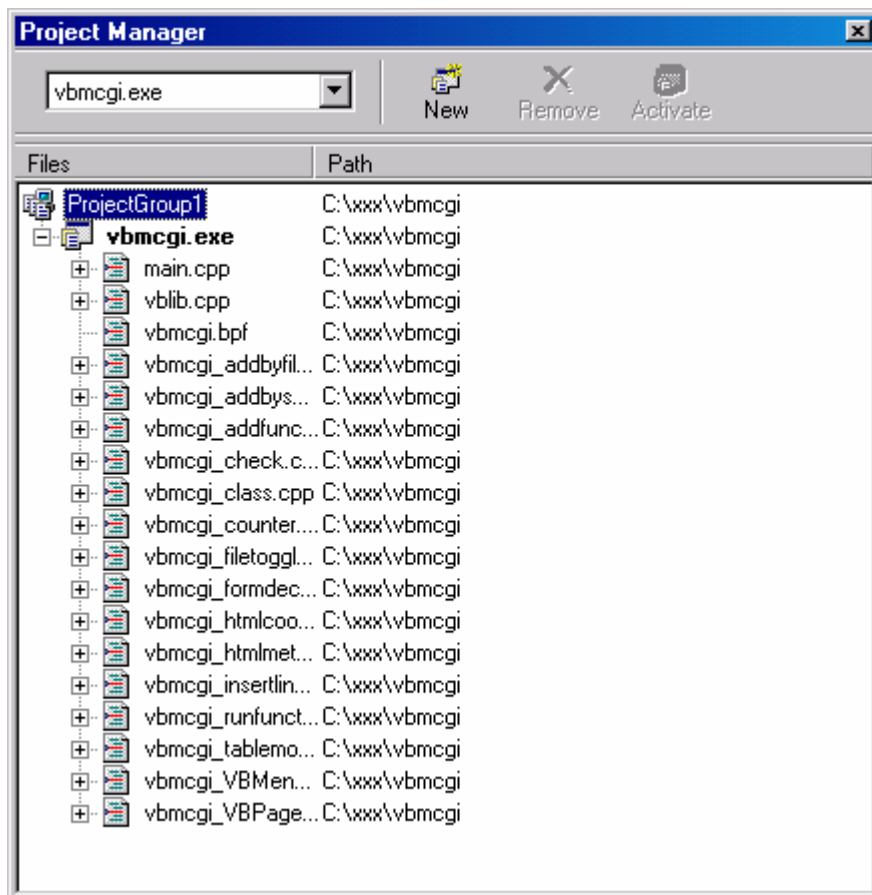
A screenshot of a Windows command window. The title bar is blue and reads "C:\xxx\vbmcgi\Debug\vbmcgi.exe". The window has standard minimize, maximize, and close buttons. The text inside the window is as follows:

```
Content-type:text/html
<html><body>
Some text in html
</body></html>
Press any key to continue_
```

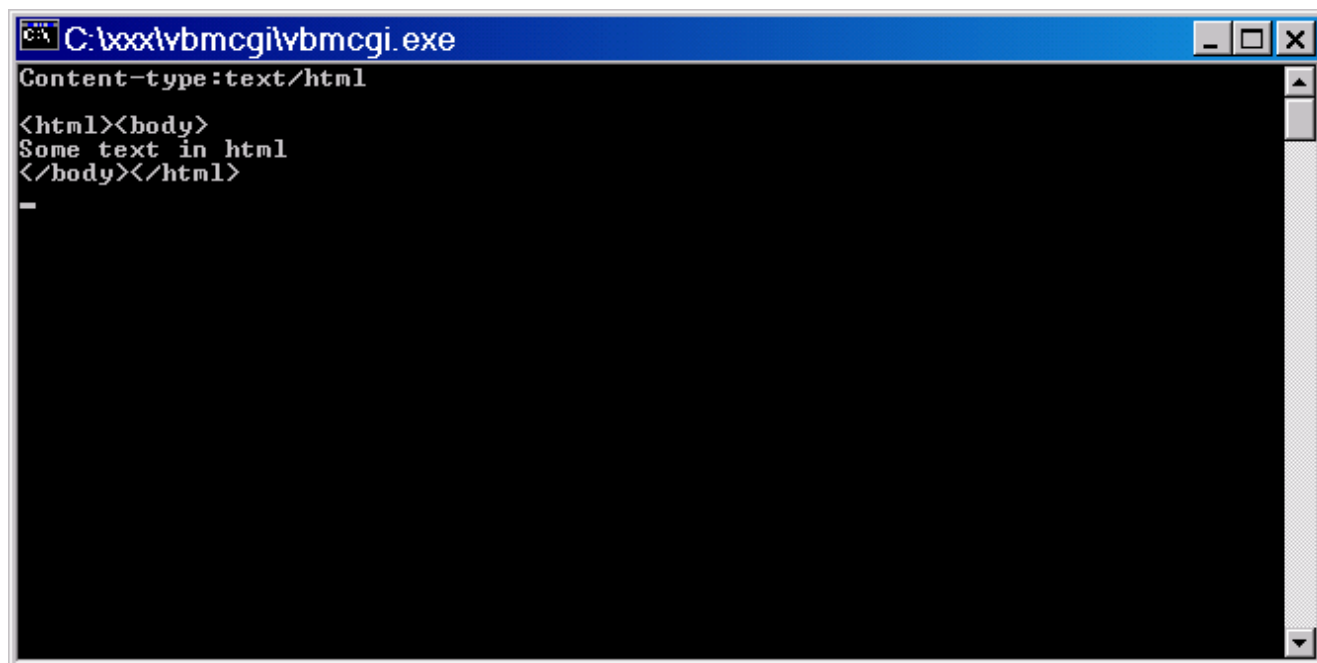
14.5.2 Windows, Borland Builder C++ 5.0



Uma forma fácil de se usar o Borland Builder com VBMcgi (testado na versão 5.0), é abrir o arquivo de projeto vbmcgi.bpr, que existe na distribuição. Por fazer isso, se um dos arquivos de fonte ocasionalmente abrir espontaneamente, ignore-o e feche-o. Apenas execute o projeto por pressionar F9. O project manager (<ctrl><alt>F11) deverá mostrar arquivos como mostrado na figura abaixo.



Por rodar o projeto que existe na distribuição, sob Borland Builder 5.0, espere ver uma saída como mostrado na figura abaixo.



14.5.3 Unix, Gnu C++ compiler

Crie um diretório de trabalho para a biblioteca VBMcgi. Suponha que esse diretório seja /users/me/vbmcgi. Copie o arquivo de distribuição vbmcgi22.zip para esse diretório, e descomprima (unzip) o arquivo. Use o arquivo script chamado make.sh, que é como mostrado abaixo. O símbolo ‘⇒’ significa que a linha continua sem ‘return’.

```
# make.sh
echo Rebuilding the VBMcgi library. see www.vbmcgi.org

echo compiling
g++ -c vllib.cpp vbmcgi_addbyfile.cpp vbmcgi_check.cpp vbmcgi_filetoggle.cpp
⇒ vbmcgi_htmlmethods.cpp vbmcgi_tablemonth.cpp vbmcgi_addbyspecialfile.cpp
⇒ vbmcgi_class.cpp vbmcgi_formdecode.cpp vbmcgi_insertline.cpp vbmcgi_VBPageCount.cpp
⇒ vbmcgi_addfunction.cpp vbmcgi_counter.cpp vbmcgi_htmlcookie.cpp vbmcgi_runfunction.cpp
⇒ vbmcgi_VBMenuHtml.cpp

echo binding lib
ar -r libvbmcgi.a vllib.o vbmcgi_addbyfile.o vbmcgi_check.o vbmcgi_filetoggle.o
⇒ vbmcgi_htmlmethods.o vbmcgi_tablemonth.o vbmcgi_addbyspecialfile.o vbmcgi_class.o
⇒ vbmcgi_formdecode.o vbmcgi_insertline.o vbmcgi_VBPageCount.o vbmcgi_addfunction.o
⇒ vbmcgi_counter.o vbmcgi_htmlcookie.o vbmcgi_runfunction.o vbmcgi_VBMenuHtml.o
```

Assegure-se que o arquivo script pode ser executado, por executar o comando abaixo.

```
chmod 755 make.sh
```

Execute o script make.sh (se . ‘diretório corrente’ não estiver no path, o comando para executar o script será ‘./make.sh’). A saída deve ser como abaixo. Após executar o script, o arquivo de biblioteca libvbmcgi.a deverá ter sido criado.

```
Rebuilding the VBMcgi library. see www.vbmcgi.org
compiling
binding lib
```

Para usar a biblioteca, crie o arquivo hello_vbmcgi.cpp como abaixo.

```
// hello_vbmcgi.cpp
#include "VBMcgi.h"
int main () {
    VBMcgi cgi;
    cgi.htmlCompleteHeader();
    cout << "<html><body>" << endl;
    cout << "Some text in html" << endl;
    cout << "</body></html>" << endl;
    return 0;
}
```

Construa (build) o arquivo hello_vbmcgi.cgi com o comando abaixo.

```
g++ hello_vbmcgi.cpp libvbmcgi.a -o hello_vbmcgi.cgi
```

O arquivo hello_vbmcgi.cgi deverá ter sido criado. Esse arquivo pode ser executado por linha de comando. Para fazê-lo, comande ‘./hello_vbmcgi.cgi’. Mova o programa cgi para o diretório cgi-bin do seu computador. Possivelmente esse diretório é /user/me/public_html/cgi-bin.

Para manter os arquivos organizados, você possivelmente quererá os arquivos fonte próprios de programas cgi num diretório separado dos arquivos da biblioteca VBMcgi. Digamos que o diretório para armazenar os arquivos fonte de programas cgi pessoais seja /user/me/cgi_src. Portanto, o arquivo hello_vbmcgi.cpp, assim como os próximos arquivos fonte de programas cgi, deverão ficar nesse diretório. É conveniente criar um arquivo script vbmcc.sh como mostrado abaixo. Assegure-se que esse arquivo script é executável com o comando “chmod 755 vbmcc.sh”. O arquivo vbmcc.sh deverá estar num diretório que tenha path para o usuário (pode ser o diretório /user/me/cgi_src directory, se esse diretório estiver adicionado ao path).

```
# vbmcc.sh
g++ $1.cpp $2 $3 $4 $5 $6 /users/me/vbmcgi/libvbmcgi.a -I/users/me/vbmcgi -o $1.cgi
```

Tendo feito isso, agora pode-se compilar programas cgi por chamar o script criado, como mostrado abaixo.

```
vbmcgi.sh hello_vbmcgi
```

Se mais de um arquivo fonte for usado num projeto de programa cgi, apenas adicione os arquivos na linha de comando, como mostrado abaixo.

```
vbmcgi.sh hello_vbmcgi extra_file_1.cpp extra_file_2.cpp extra_file_3.cpp
```

14.6 Usando VBMcgi

Para ler essa seção, é assumido que o leitor já fez o seguinte:

- Baixou (download) o arquivo de distribuição da VBMcgi.
- Instalou a biblioteca VBMcgi.
- Compilou e executou o exemplo hello_vbmcgi, e o executou em linha de comando (mas não ainda como programa CGI).

14.6.1 Programas CGI e web server

Para fazer programas CGI serem executados, o web server precisa estar sendo executado. Um navegador pode abrir arquivos html e outros tipos sem a necessidade do web server. Mas os programas CGI somente podem ser executados se forem chamados pelo web server. Por isso, é necessário que um software web server esteja instalado no seu computador. O Apache é de longe o programa mais usado como web server, com versões para Windows e Unix. Trata-se de um software gratuito, maduro, poderoso, estável e rápido. Algumas versões de Windows vem com o web server da Microsoft chamado IIS. Esse servidor também pode ser usado, ou pode ser substituído por Apache para Windows. Excetuando-se o Apache e o IIS, poucos programas de web server são realmente usados.

Para um computador com um web server instalado, há geralmente um diretório no computador mapeado para a web com um certo URL. Por exemplo, o diretório \users\me\public_html é mapeado pelo web server para que seja

acessado pelo URL `http://www.one_name.com`. Portanto, se alguém apontar o browser para `http://www.one_name.com/one_file.html`, o web server vai copiar o conteúdo de `one_file.html` para o cliente. Se nenhum arquivo for especificado (`http://www.one_name.com/`), então um arquivo implícito (*default*) é enviado. Um nome usual para esse arquivo implícito é “`index.html`”. Arquivos podem ser referenciados de forma relativa com relação ao nome do diretório e a URL, isto é, um diretório no computador torna-se um diretório na URL.

Quando desenvolvendo software para web com CGI, geralmente se usa um computador cliente sem conexão com a Internet. Isso é possível a partir da instalação de um web server e colocando os programas CGI em diretório apropriado (debaixo do diretório mapeado pelo web server), e acessando-o localmente como `http://localhost/mycgi.cgi`. Por usar esse método, não há qualquer dependência de conexão externa.

A forma mais simples de chamar programas CGI é chama-los diretamente pelo web server, como se fossem páginas html normais. O que o programa CGI coloca no console é enviado para o browser do cliente. O protocolo http requer que o programa CGI envie um pequeno cabeçalho (*header*) antes do envio dos dados propriamente ditos. Há vários tipos de cabeçalho. O mais comum é o cabeçalho para dados html, como mostrado abaixo (atenção para a existência de duas letras <fim-de-linha> consecutivas após a sentença “`Content-type:text/html`”).

```
Content-type:text/html
```

Após o cabeçalho de dados html ser enviado para o console pelo programa CGI, o que quer que seja enviado para o console é diretamente enviado para o navegador. O cliente recebe dados html como se fosse um arquivo html enviado diretamente pelo web server. Portanto, outros arquivos podem ser referenciados (como `*.jpg`, `*.gif`, java applet, flash, etc.) da mesma forma como o html referencia esses arquivos. Se o cliente escolher ver o código fonte de um programa CGI a partir do navegador, o cabeçalho de dados html não será visto. Apenas o que se segue após o cabeçalho será visto.

Executando-se em linha de comando o programa “`hello_cgi.cgi`”, a saída no console é mostrada abaixo.

```
Content-type:text/html
```

```
<html><body>
Some text in html
</body></html>
```

Uma vez mais vale lembrar que um programa CGI pode ser feito com QUALQUER linguagem de computador, seja compilada ou interpretada. E assim o é em função das opções de projeto do time original²⁰ que projetou essa estrutura.

Suponha que o programa `hello.cgi.cgi`²⁵ é um executável binário construído (*built*) a partir do arquivo fonte `hello.cgi.cpp` abaixo.

```
// hello.cgi.cpp
#include <iostream.h>
int main () {
    cout << "Content-type:text/html" << endl << endl;
    cout << "<html><body>" << endl;
    cout << "Some text in html" << endl;
    cout << "</body></html>" << endl;
    return 0;
}
```

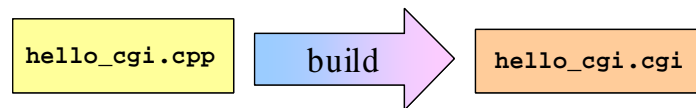
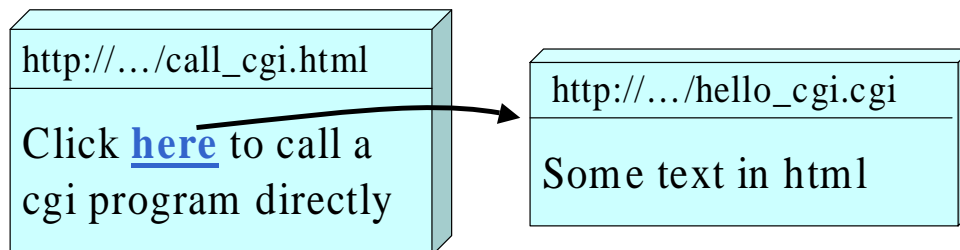


Figura 37: Construindo um programa em C++
(o programa CGI é um programa comum em linha de comando)

Uma página html simples que se liga a um programa CGI é mostrada abaixo.

```
<!-- simple_call_to.cgi.html -->
<html><body>
Click <a href="hello.cgi.cgi">here</a> to see the result of the CGI program.
</body></html>
```

O efeito na web é mostrado como abaixo.



Uma outra forma simples de se chamar um programa CGI por uma página html é feito pelo uso do tag de form (formulário). Veja o tutorial VBMcgi em no url abaixo.

http://www.vbmcgi.org/index.chtml?s_p=hello_vbmcgi.html²⁶

```
<!-- form_call_to.cgi.html -->
<html><body>
Hello CGI example: <p>
<form action="hello.cgi.cgi" method="get">
<input type="submit" value="send">
</form>
</body></html>
```

²⁵ Extensões *.cgi podem ser feitas executáveis tanto em Unix quanto em Windows.

²⁶ Na página www.vbmcgi.org, os programas cgi programs tem extensão *.chtml, além de *.cgi.

O programa CGI é um programa comum em linha de comando. O exemplo simples abaixo mostra um programa CGI executando um laço de programação. Acesso a arquivos e programas de banco de dados também são permitidos em programas CGI²⁷.

```
// multiply_table.cpp (tabuada)
#include <iostream.h>
int main () {
    cout << "Content-type:text/html" << endl << endl;
    cout << "<html><body>" << endl;
    int multiply_table = 5;
    int i;
    for (int i=1 ; i <= 10 ; i++)
        cout << i* multiply_table << "<br>" << endl;
    cout << " </body></html>" << endl;
    return 0;
}
```

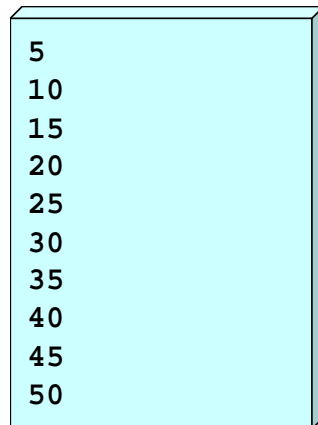
A saída do programa multiply_table.cgi é mostrada abaixo.

```
Content-type:text/html
```

```
<html><body>
5<br>
10<br>
15<br>
20<br>
25<br>
30<br>
35<br>
40<br>
45<br>
50<br>
</body></html>
```

No navegador do cliente, a saída tem o aspecto mostrado abaixo. Veja o exemplo tutorial da VBMcgi em http://www.vbmcgi.org/index.shtml?s_p=multiply_table.html.

²⁷ Mas há que se considerar as restrições de acesso. Em alguns casos, em unix, os programas CGI são executados com identificação de “nobody”, ou de “www”, ao invés da identificação do próprio usuário. Portanto, devido a restrições de acesso do diretório, o programa CGI pode não ter o direito de escrever num arquivo no diretório do usuário, a menos que esse diretório tenha direito de acesso liberado para o mundo (algo perigoso de se fazer). Se a opção “su_exec” do web server estiver ativada, o programa CGI é executado com a identificação do próprio usuário, e portanto o problema é resolvido. Problemas semelhantes podem ocorrer em Windows.



5
10
15
20
25
30
35
40
45
50

14.6.2 Programas CGI e o ambiente (environment)

O protocolo http permite a comunicação de páginas em html com programas pela interface CGI, usando o ambiente (*environment*). Por exemplo: o nome das variáveis de um formulário html e seu conteúdo são armazenado em variáveis do ambiente. O “ambiente” é uma entidade que armazena variáveis, que são visíveis por quaisquer programas. O “path” (caminho) é uma das famosas variáveis de ambiente. Programas podem ler e escrever em variáveis de ambiente.

Em C, usando a função da biblioteca padrão “getenv”, uma variável de ambiente pode ser lida. Algumas das variáveis de ambiente que tem utilidade para programas cgi, podem ser lidas e exibidas pelo programa abaixo. Nesse programa, para facilitar, fez-se uma função printEnv, que exibe o nome e conteúdo de uma variável de ambiente. As variáveis de ambiente retornam em formato string.

```
// getenv.cpp

#include <iostream.h> // cout
#include <stdlib.h> // getenv

void printEnv(const char *var) {
    char *t = getenv(var);
    if (!t) t=""; // never reference a pointer to zero
    cout << "Environment[" << var << "] = \"" << t << "\"<br>" << endl;
}

int main () {
    cout << "Content-type: text/html" << endl << endl; // html header
    printEnv("SERVER_SOFTWARE");
    printEnv("SERVER_NAME");
    printEnv("GATEWAY_INTERFACE");
    printEnv("SERVER_PROTOCOL");
    printEnv("SERVER_PORT");
    printEnv("REQUEST_METHOD");
    printEnv("HTTP_ACCEPT");
    printEnv("PATH_INFO");
    printEnv("PATH_TRANSLATED");
    printEnv("SCRIPT_NAME");
    printEnv("QUERY_STRING");
    printEnv("REMOTE_HOST");
}
```

```

printEnv("REMOTE_ADDR");
printEnv("AUTH_TYPE");
printEnv("REMOTE_USER");
printEnv("REMOTE_IDENT");
printEnv("CONTENT_TYPE");
printEnv("CONTENT_LENGTH");
printEnv("HTTP_USER_AGENT");
printEnv("HTTP_REFERER");
return 0;
}

```

Se o programa getenv.cgi for executado em linha de comando, a saída deverá ser como abaixo (todas as variáveis vazias).

Content-type: text/html

```

Environment[SERVER_SOFTWARE] = ""<br>
Environment[SERVER_NAME] = ""<br>
Environment[GATEWAY_INTERFACE] = ""<br>
Environment[SERVER_PROTOCOL] = ""<br>
Environment[SERVER_PORT] = ""<br>
Environment[REQUEST_METHOD] = ""<br>
Environment[HTTP_ACCEPT] = ""<br>
Environment[PATH_INFO] = ""<br>
Environment[PATH_TRANSLATED] = ""<br>
Environment[SCRIPT_NAME] = ""<br>
Environment[QUERY_STRING] = ""<br>
Environment[REMOTE_HOST] = ""<br>
Environment[REMOTE_ADDR] = ""<br>
Environment[AUTH_TYPE] = ""<br>
Environment[REMOTE_USER] = ""<br>
Environment[REMOTE_IDENT] = ""<br>
Environment[CONTENT_TYPE] = ""<br>
Environment[CONTENT_LENGTH] = ""<br>
Environment[HTTP_USER_AGENT] = ""<br>
Environment[HTTP_REFERER] = ""<br>

```

Mas se esse mesmo programa for executado não pelo usuário, mas pelo web server como um programa cgi, o servidor web salva todos em variáveis de ambiente antes de chamar o programa cgi. Portanto, a saída (vista pelo browser) será mais ou menos como mostrado abaixo (algumas variáveis não estão vazias). Veja o exemplo tutorial da VBMcgi em

http://www.vbmcgi.org/index.chtml?s_p=environment.html.

URL: <http://www.vbmcgi.org/getenv.cgi>

```
Environment[SERVER_SOFTWARE] = "Apache/1.3.12 Ben-SSL/1.41 (Unix) tomcat/1.0"
Environment[SERVER_NAME] = "vbmcgi.org"
Environment[GATEWAY_INTERFACE] = "CGI/1.1"
Environment[SERVER_PROTOCOL] = "HTTP/1.1"
Environment[SERVER_PORT] = "80"
Environment[REQUEST_METHOD] = "GET"
Environment[HTTP_ACCEPT] = "image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
    application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */*"
Environment[PATH_INFO] = ""
Environment[PATH_TRANSLATED] = ""
Environment[SCRIPT_NAME] = "/getenv.cgi"
Environment[QUERY_STRING] = ""
Environment[REMOTE_HOST] = ""
Environment[REMOTE_ADDR] = "146.164.50.16"
Environment[AUTH_TYPE] = ""
Environment[REMOTE_USER] = ""
Environment[REMOTE_IDENT] = ""
Environment[CONTENT_TYPE] = ""
Environment[CONTENT_LENGTH] = ""
Environment[HTTP_USER_AGENT] = "Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)"
Environment[HTTP_REFERER] = ""
```

Há muita informação útil em variáveis de ambiente. Mas esse assunto não será explorado no momento.

Uma variável de ambiente de interesse particular é a `QUERY_STRING`. Essa variável armazena o conteúdo das variáveis do formulário html (seus nomes e conteúdos). Nessa variável, em uma única string todas os nomes de variável e seus conteúdos são armazenados. Para evitar problemas com caracteres não padrão, uma “seqüência scape” especial é usada. O padrão definido para o formato de `query_string` foi projetado para que se possa decodifica-la de forma completa, para um conteúdo de propósito genérico. Ou seja, é possível escrever um programa que decodifica a `query_string` e recupera a informação das variáveis e seus conteúdos. É óbvio que é tarefa de uma biblioteca para apoiar a programação cgi decodificar as variáveis do formulário (o que inclui reverter a seqüência scape), e prover acesso fácil às variáveis do formulário e seus conteúdos.

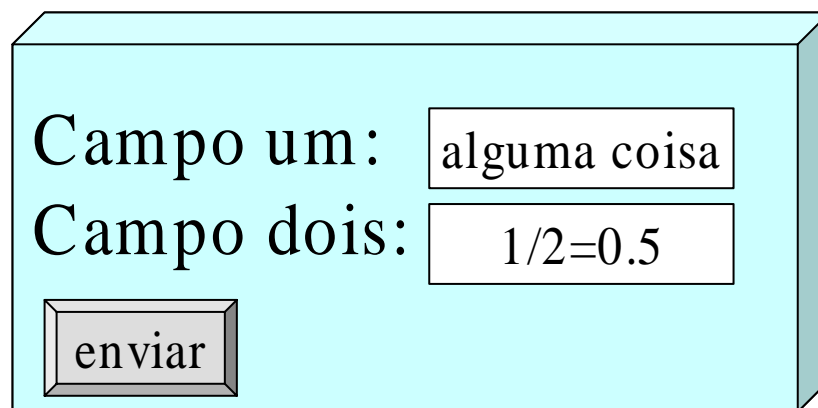
Há dois métodos de chamada de um programa cgi a partir da tag `<form>` - get e post. Usando o método get, o conteúdo da `query_string` é mostrado na url, após o ‘?’, usando a seqüência como mostrado abaixo.

`campo_1_nome= campo_1_conteúdo& campo_2_nome= campo_2_conteúdo ...`

O nome do campo e o conteúdo é mostrado sob código de sequência scape. No exemplo abaixo, uma página html simples tem um formulário com 2 campos, usando o método get.

```
<!-- test_getenv.html -->
<html><body>
<form action="getenv.cgi" method="get">
Campo um: <input type="text" name="campo_um" size="10"><br>
Campo dois: <input type="text" name="campo_dois" size="10"><br>
<input type="submit" value="enviar">
</form>
</body></html>
```

Se o conteúdo dos campos for respectivamente “alguma coisa” e “1/2=0.5”, o aspecto no navegador será como mostrado abaixo.



Quando o usuário clica no botão “enviar”, o programa getenv.cgi será chamado. O campo URL que chama o programa getenv.cgi será como mostrado abaixo. A sequência escape é usada para representar os caracteres especiais do campo_dois.

URL:

`http://***/getenv.cgi?campo_um=alguma_coisa&campo_dois=1%2F2%3D0.5.`

Caso se use o método put ao invés do método get, então os nomes dos campos do formulário e o seu conteúdo são passados para a query_string pelo console, e não pelo URL. Portanto nesse caso a URL não mostra o conteúdo da query_string. Uma utilidade do método put é para o caso de existência de conteúdo muito longo (como é o caso no uso de text_area). O método get tem limite de 256 caracteres, enquanto o método put não tem limite de conteúdo.

Se usando a biblioteca VBMcgi, é transparente para o programador se a página html que chamou o programa cgi usou método put ou get.

Pode-se fazer testes interessantes com a sequência escape com o código simples abaixo. Experimente esse programa, trocando a string de entrada por outras que se queira testar.

```
// code to test the scape sequence
#include "vbmcgi.h"
```

```
int main () {
    VBString a;
    a = "1/2=0.5";
    cout << a << endl;
    a.scapeSequence();
    cout << a << endl;
    a.scapeSequenceReverse();
    cout << a << endl;
    return 0;
}
```

14.6.3 Hello VBMcgi

Se a VBMcgi foi instalada com sucesso no seu sistema, então você deverá compilar programas em C++ com essa biblioteca. O arquivo `hello_vbmcgi.cpp` abaixo é um exemplo muito simples que usa a VBMcgi. Esse exemplo serve para testar se a VBMcgi está instalada corretamente.

```
// hello_vbmcgi.cpp
#include "vbmcgi.h"
int main () {
    VBMcgi cgi;
    cgi.htmlCompleteHeader();
    cout << "<html><body>" << endl;
    cout << "Some text in html" << endl;
    cout << "</body></html>" << endl;
    return 0;
}
```

O arquivo `hello_vbmcgi.cpp` inclui o cabeçalho (*header file*) principal da VBMcgi, que é o `vbmcgi.h`, contendo todas as definições necessárias. Esse cabeçalho também inclui alguns cabeçalhos padrão do C++ (como o `iostream.h`), portanto não é necessário incluí-lo explicitamente. Há uma classe declarada no arquivo “`vbmcgi.h`”, que se chama `vbmcgi`. Instancia-se um objeto dessa classe com o nome “`cgi`”. Nesse exemplo simples, apenas um método é chamado do objeto `cgi`. O método é “`htmlCompleteHeader`”, que simplesmente envia o cabeçalho html para o console. Após o cabeçalho html, o que se segue são dados html, como no exemplo `hello_cgi.cpp`. Se fato, a saída de `hello_vbmcgi.cpp` é a mesma da `hello_cgi.cpp`.

14.6.4 Decodificando o formulário

A biblioteca VBMcgi provê suporte para decodificação de variáveis html por chamar o método “`formDecode`” a partir de um objeto do tipo “`vbmcgi`”. Esse método lê a `query_string` do ambiente, reverte a sequência escape, decodifica completamente as variáveis do formulário e armazena o resultado numa lista encadeada (escondida do programador). Para acessar às variáveis e seu conteúdo, o programador chama o método “`getVarContents`” (necessariamente após ter chamado uma vez o método `formDecode`).

Ressalte-se que a forma segura de se processar strings em C++ é NÃO manipular diretamente array of char. Ao invés disso, é fortemente recomendado que se use classe de string para essa

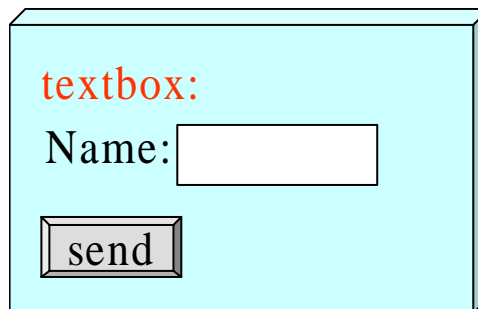
finalidade. Na VBMcgi há uma classe de string portátil e de propósito genérico chamada VBString.

Veja detalhes do uso de VBString em <http://www.vbmcgi.org/vbllib/>, ou nesse mesmo livro.

14.6.4.1 textbox (caixa de texto)

O código html abaixo produz um formulário com um campo textbox.

```
<html><body>
textbox:<br>
<form action="formdecode.cgi" method="get">
Name:<input type="text" name="fieldname" size="20"><br>
<input type="submit" value="send">
</form>
</body></html>
```



Um programa cgi para decodificar textbox, obtendo como string o conteúdo da variável, é mostrado abaixo. Como o conteúdo da variável é sempre uma string, caso o seu significado seja numérico, é preciso chamar uma função que converta a string em número (por exemplo int, ou double).

```
// formdecode.cpp
#include "vbmcgi.h"
int main () {
    VBMcgi cgi;
    cgi.formDecode(); // decode form variables
    VBString str = cgi.getVarContent("fieldname");
    // continue cgi code
    return 0;
}
```

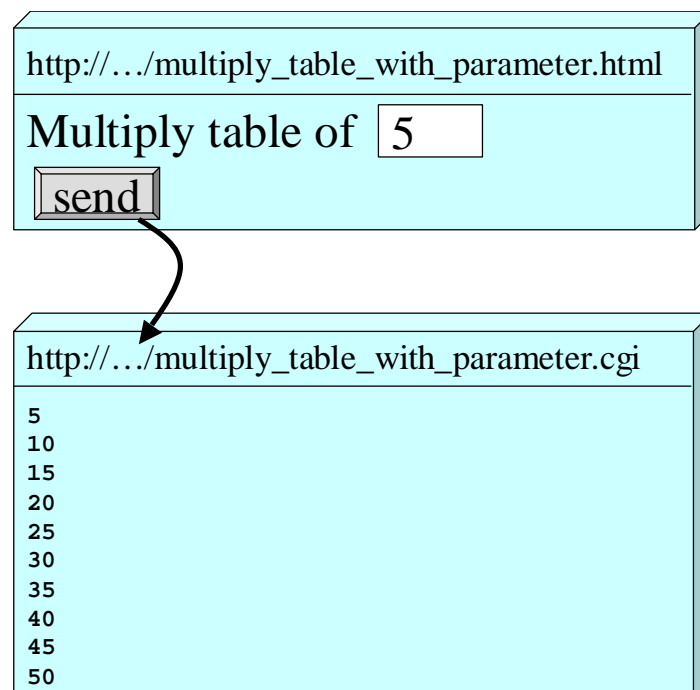
Outro exemplo de decodificação das variáveis do formulário com textbox é o caso de tabuada (*multiply table*) com um parâmetro, como mostrado abaixo.

```
<!-- multiply_table_with_parameter.html -->
<html><body>
<form action="multiply_table_with_parameter.cgi" method="get">
Multiply table of <input type="text" name="mtable" size="3"><br>
<input type="submit" value="send">
</form> </body></html>
```

O arquivo fonte de programa cgi para processar essa página html é mostrado abaixo. Nesse código, as variáveis do formulário são decodificadas pelo método “formDecode”. O conteúdo do campo “mtable” do formulário html é retornado pelo método “getVarContent” com “mtable” como parâmetro. Um objeto string (VbString) “mt_str” recebe o conteúdo de “mtable”. Nesse exemplo, o retorno seria “5” (uma string contendo “5”, não o inteiro 5). Pode-se converter a string “5” para o inteiro 5 com a função da biblioteca padrão “atoi”. Após isso, dados html são enviados para o console, com um laço baseado em for para enviar a tabuada para o navegador do usuário.

```
// multiply_table_with_parameter.cpp
#include "vbmcgi.h"
int main () {
    VBMcgi cgi;
    cgi.htmlCompleteHeader();
    cgi.formDecode(); // decode form vars
    VbString mt_str = cgi.getVarContent("mtable"); // get mtable as string
    int multiply_table = atoi(mt_str); // convert to int
    cout << "<html><body>" << endl;
    for (int i=1 ; i <= 10 ; i++)
        cout << i*multiply_table << "<br>" << endl;
    cout << "</body></html>" << endl;
    return 0;
}
```

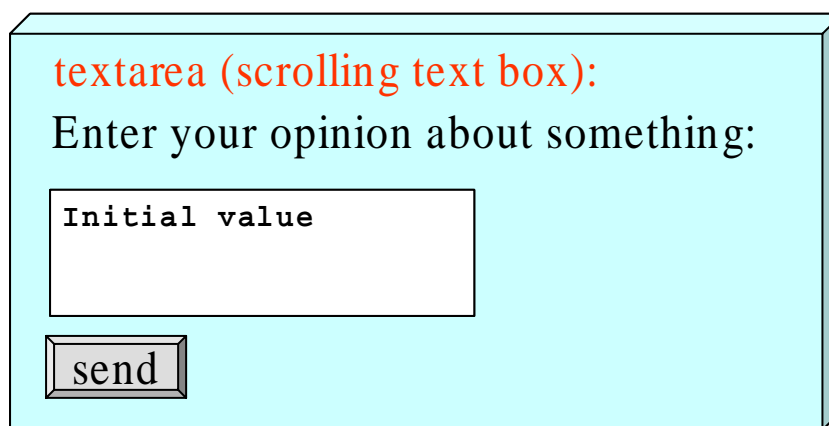
O resultado no navegador será como mostrado abaixo. Veja o exemplo tutorial da VBMcgi com título “Multiply table with parameter” (tabuada com parâmetro), e experimente outros valores (diferente de 5) no formulário.



14.6.4.2 textarea (área de texto).

O código html abaixo produz um formulário com um campo textarea. “Enter your opinion about something” significa “Entre sua opinião sobre alguma coisa”.

```
<html><body>
textarea (scrolling text box):<br>
<form action="formdecode.cgi" method="get">
Enter your opinion about something:<br>
<textarea name="opinion" rows="4" cols="55">Initial value</textarea>
<input type="submit" value="send">
</form>
</body></html>
```



Um programa cgi para decodificar textarea, obtendo como string o conteúdo da variável, é mostrado abaixo. O exemplo é notavelmente semelhante ao caso de textbox. O conteúdo da variável de retorno pode ser muito grande. A variável do tipo string (no caso VBString) pode receber conteúdo de qualquer tamanho. Caso o conteúdo possua várias linhas, ainda assim todo o conteúdo será armazenado numa única string, sendo que o char ‘\n’ (return) ocorrerá entre as linhas. Cabe ao programador tratar o conteúdo dessa variável e usa-lo como fizer sentido.

```
// formdecode.cpp
#include "vbmcgi.h"
int main () {
    VBMcgi cgi;
    cgi.formDecode(); // decode form variables
    VBString str = cgi.getVarContent("opinion");
    // continue cgi code
    return 0;
}
```

14.6.4.3 checkbox (caixa de checagem)

O código html abaixo produz um formulário com campos checkbox. “You like” significa “Você gosta de”; “Sports, Books, News” significam “Esporte, Livros, Notícias”.

```
<html><body>
checkbox:<br>
<form action="formdecode.cgi" method="get">
You like:<br>
<input type="checkbox" name="like_sports" checked value="ON">Sports<br>
```



```



```

Existe um método na VBMcgi específico para decodificar o checkbox, chamado `getCheckBox`. O retorno desse método é booleano verdadeiro caso exista um “check” no checkbox, e falso no caso de não existir. O programa abaixo é um exemplo de decodificação de checkbox.

```

// formdecode.cpp
#include "vbmcgi.h"
int main () {
    VBMcgi cgi;
    cgi.formDecode(); // decode form variables
    bool b_sports = cgi.getCheckBox("like_sports");
    bool b_books  = cgi.getCheckBox("like_books");
    bool b_news   = cgi.getCheckBox("like_news");
    // continue cgi code
    return 0;
}

```

14.6.4.4 radio button (botão radial)

O radio button num formulário é um item de um grupamento, em que a seleção de qualquer dos itens desse grupamento implica na de-seleção dos demais. É o tipo de dados que se usaria para obter a resposta de uma questão de uma prova de múltipla escolha, por exemplo. Um exemplo de código html que gera radio button é mostrado abaixo. Opcionalmente pode-se pré marcar uma das opções do radio button, como mostrado no exemplo com a opção “study”.

As quatro opções do exemplo estão no grupamento “after_dinner” (após o jantar). Cada opção tem um valor, que é uma string. É essa a string que retorna da leitura do formulário. A string da opção não é a string que o usuário vê na tela. Para facilitar a manipulação dos dados, atribuiu-se strings de valor nas opções que são na verdade números. Com isso, a conversão para número torna-se imediata.

“After dinner you” significa “Após o jantar você”; “brush your teeth” significa “escova os dentes”; “watch TV” significa “assiste TV”; “study” significa “estudar”; “read a book” significa “lê um livro”.

```
<html><body>
radio button:<br>
<form action="formdecode.cgi" method="get">
After dinner you:<br>
<input type="radio" name="after_dinner" value="1">Brush your teeth<br>
<input type="radio" name="after_dinner" value="2">Watch TV<br>
<input type="radio" name="after_dinner" value="3" checked>Study<br>
<input type="radio" name="after_dinner" value="4">Read a book
</form>
</body></html>
```



Um programa cgi para decodificar o radio button é mostrado abaixo. O retorno do método `getVarContent` é do tipo `string`, e corresponde a string de valor da opção que se selecionou. No caso do exemplo, as strings possíveis são “1”, “2”, “3” e “4”. Essas strings de valor foram escolhidas para facilitar a conversão para inteiro com a função global da biblioteca padrão chamada “atoi” (ascii to integer). Para exemplificar o uso da seleção de radio button, criou-se um array de strings carregado com constantes. Por referenciar esse array, se está retornando uma das strings do array. O programa abaixo gera na saída uma string relacionada ao radio button selecionado.

```
// formdecode.cpp
#include "vbmcgi.h"
int main () {
    VBMcgi cgi;
    cgi.formDecode(); // decode form variables
    VBString n_str = cgi.getVarContent("after_dinner");
    int n = atoi(n_str);
    const char *stringsAfterDinner[4] = {
        "brush the teeth",
        "give TV a good watch",
        "go study for a while",
        "read the favorite book"
    }
```

```

};
// continue cgi code
cgi.htmlCompleteHeader();
cout << stringsAfterDinner[n-1] << endl;
return 0;
}

```

14.6.4.5 drop-down (deixar escapulir para baixo)

Uma outra alternativa de obter uma seleção de uma lista é usar o tipo drop-down, também conhecido com list-menu. Esse tipo de dados, de forma semelhante ao radio button, possui uma string interna para cada opção. Essa string não é exibida ao usuário final, e é a string de retorno quando se pega o conteúdo da variável.

No exemplo abaixo, pergunta-se a cor favorita do usuário. O retorno é “1”, “2” ou “3”, pela mesma razão que no exemplo de radio button.

```

<html><body>
drow down:<br>
<form action="formdecode.cgi" method="get">
Your favorite color is: <br>
<select name="favorite_color" size="1">
<option value="1">Green</option>
<option value="2">Red</option>
<option value="3">Blue</option>
</select>
</form>
</body></html>

```

drop down:

Your favorite color is:

Green ▼

Green

Red

Blue

send

O código de um programa cgi que decodifica drop down é mostrado abaixo. Para exemplificar um mínimo de processamento com o resultado da variável que o usuário seleciona, exibe-se uma string relacionada ao que foi selecionado.

```

// formdecode.cpp
#include "vbmcgi.h"
int main () {
    VBMcgi cgi;
    cgi.formDecode(); // decode form variables
}

```

```

VBString n_str = cgi.getVarContent("favorite_color");
int n = atoi(n_str);
const char *stringsForColor[3] = {
    "the color green",
    "red, a hot color",
    "blue, the color of the sky"
};
// continue cgi code
cgi.htmlCompleteHeader();
cout << stringsForColor[n-1] << endl;
return 0;
}

```

14.6.4.6 Exemplo completo na web

Os exemplos do livro podem ser vistos em funcionamento no link abaixo.

http://www.vbmcgi.org/index.chtml?s_p=complete_form_reference.html

14.6.5 Usando a funcionalidade “string change”

A funcionalidade “string change” (mudança de palavra) é uma das funcionalidades da biblioteca VBMcgi permite o “isolamento entre o webmaster e o webdesigner” - a regra de ouro da VBMcgi, a característica que dá distinção a essa biblioteca.

Essa funcionalidade faz com que se possa gerar saída html no console a partir de um arquivo html externo. Esse arquivo html pode ser gerado a partir de qualquer software comercial que os webdesigners geralmente gostam de usar (por exemplo: Front Page, Dream Weaver, Go Live, etc.). A interatividade é obtida por “ensinar” pares de strings para o objeto vbmcgi. Um par de strings contém uma string para “procurar”, e outra para “trocar”. Um número ilimitado de strings pode ser “ensinadas” para o objeto vbmcgi.

Veja agora o código fonte de string_change_feature.cpp, mostrado abaixo. Por compilar esse código e ligar juntamente com a biblioteca VBMcgi, é gerado o arquivo executável string_change_feature.cgi²⁸. Nesse exemplo, após instanciar-se o objeto vbmcgi chamado “cgi”, 4 pares de strings são “ensinados” para o objeto vbmcgi usando o método “addBySource”. O primeiro argumento é a string de procura, e o segundo argumento é a string de troca. A última linha chama o método “out”, que tem como parâmetro o nome do arquivo person.html, mostrado abaixo. O que o método “out” faz é abrir o arquivo recebido como parâmetro, ler linha a linha e enviar a saída para o console de

²⁸ Novamente deve-se lembrar que apesar de usuários de Windows terem expectativa de que apenas arquivos *.exe sejam executáveis, os arquivos *.cgi podem ser executáveis mesmo em Windows se chamados por um web server. Em unix, arquivos de qualquer extensão podem ser executáveis. Opcionalmente, pode-se usar extensão *.exe para os programas cgi que são executados em Windows, mas isso não é uma requisição.

saída. Mas antes de enviar a linha, executa todas as trocas de string “search” por “replace” que tem na sua lista.

```
// string_change_feature.cpp
#include "vbmcgi.h"
int main () {
    vbmcgi cgi; // one instance of vbmcgi object

    // add some string exchanges
    cgi.addBySource("s_name","Sergio");
    cgi.addBySource("s_telephone","222-3344");
    cgi.addBySource("s_zip","20.234-140");
    cgi.addBySource("s_email","villas@kmail.com.br");

    // open html file, execute string changes and put to browser
    cgi.out("person.html");
    return 0;
}
```

O arquivo person.html é mostrado abaixo.

```
<!-- person.html -->
<!-- This could be a sophisticated html file from html design software -->
<html><body>
Here is the person from the database:<p>
Name is: <b>s_name</b><br>
Telephone is: <b>s_telephone</b><br>
ZIP is: <b>s_zip</b><br>
email is: <b>s_email</b><p>
Thanks for using our system
</body></html>
```

A figura abaixo ilustra a funcionalidade de string change em ação. Nesse caso, uma página simples contém um link para chamar o programa string_change_feature.cgi. Um detalhe interessante é que o arquivo person.html nesse caso está no mesmo diretório de string_change_feature.cgi, portanto em princípio se poderia ver o arquivo person.html diretamente no navegador. Contudo, no caso geral, o arquivo html que é parâmetro do método “out” não precisa estar nesse diretório. A rigor, não precisa sequer estar em algum diretório debaixo de “public_html”, ou seja, os arquivos html de argumento do método “out” não precisam estar na web.

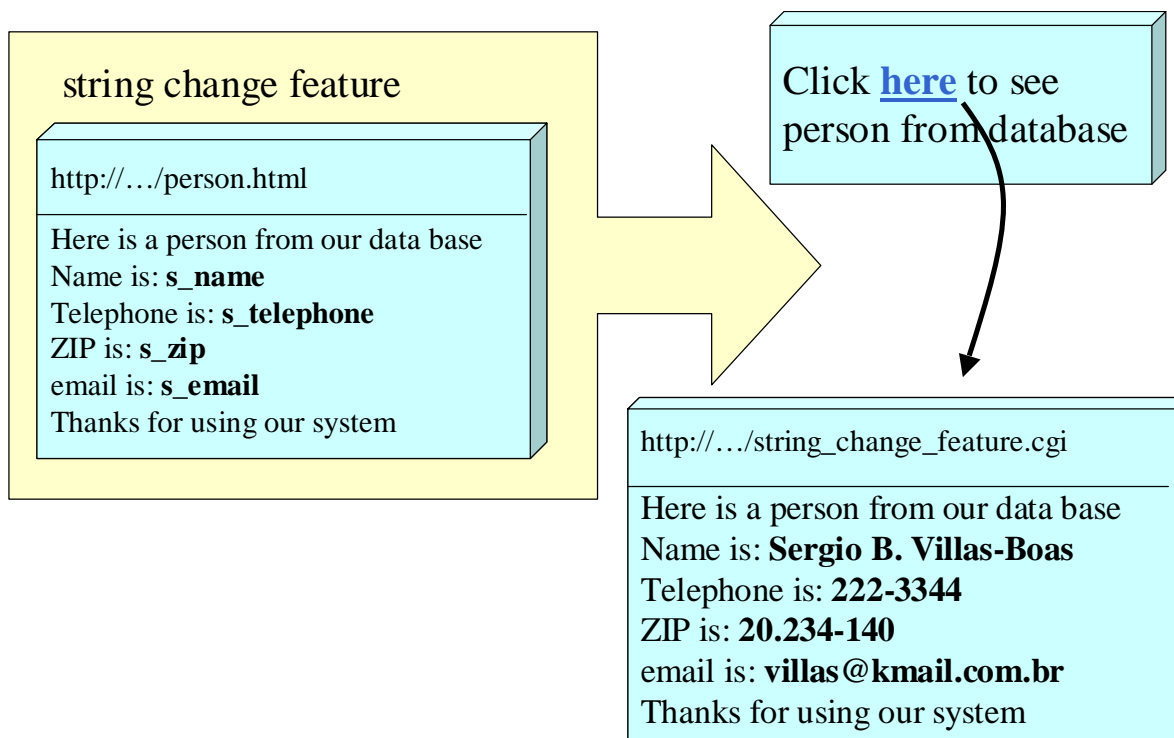


Figura 38: “string change feature”

Esse exemplo mostra como se obtém o isolamento entre o webmaster e o webdesigner, a partir da funcionalidade “string change”. O webdesigner pode produzir um código html muito sofisticado, usando sua ferramenta favorita. Tudo o que o webmaster precisa saber é o nome do arquivo, para colocar como argumento do método “out”. A complexidade do trabalho do webdesigner não afeta o trabalho do webmaster de forma alguma, nem vice-versa. O webdesigner pode mudar o arquivo person.html, e sem recompilar o programa cgi (isso é, sem precisar sequer avisar ao webmaster), o usuário final percebe a diferença. Analogamente, o webmaster pode fazer alterações no programa cgi sem precisar avisar ao webdesigner.

Um desenvolvedor para web com criatividade pode imaginar enormes possibilidades de uso da funcionalidade “string change”. Menciono uma: que tal uma página de notícias na web, em que conste uma string de data-hora no topo da página, antes das notícias propriamente ditas ? O webmaster pode escrever facilmente um programa que leia a data-hora do sistema e retorne uma string com o resultado. Coloca-se essa string no “replace” do string change, sendo que o “search” é uma string de procura como por exemplo “s_date_time”. Tudo o que o webmaster precisa saber é qual é essa string de search, para coloca-la no lugar em que a data e hora aparecerão na visualização final. O conteúdo da página web em html pode mudar livremente sem necessidade de se recompilar o programa cgi. De fato, o webmaster nem precisa saber qualquer conhecimento

de C++, VBMcgi, ou qualquer programação. Ele pode concentrar-se no seu trabalho de webdesign.

Veja o exemplo tutorial “string change feature”, que está no link abaixo.

http://www.vbmcgi.org/index.shtml?s_p=string_change_feature.html

Veja também o exemplo getDateTime do “advanced tutorial”, no link abaixo.

http://www.vbmcgi.org/index.shtml?s_p=getDateTime.html

A funcionalidade “string change” é apenas um código “cru” que é executado rapidamente, desde a primeira “search string” até a última. Não há qualquer inteligência no método em si de troca de strings. É melhor que seja assim, pois dessa forma o webmaster pode explorar livremente o uso dos recursos dessa funcionalidade com sua própria programação. As strings a serem procuradas e substituídas podem ser quaisquer. Mas se o webmaster escolher strings muito usuais para “search”, o que pode ocorrer é que todas as ocorrências dessa string serão trocadas cegamente pelo método “out”, e o resultado pode ser indesejado. Portanto, em geral convém escolher seqüências improváveis de letras para serem usadas como “search”.

14.6.5.1 Adicionando implicitamente strings

Suponha que se queira fazer um sistema simples, para apenas exibir os campos que o usuário digitou. Pode-se acrescentar implicitamente os pares de string do formulário html (nome da variável do formulário e seu conteúdo) como entrada para a funcionalidade “string change”. Para isso, basta acrescentar o a constante booleano “true” como parâmetro do método formDecode, como mostrado no link abaixo. No exemplo abaixo, um formulário possui 4 campos do tipo textbox.

```
<form method="get" action="formdecode.cgi">
<table border="0">
<tr><td align="right">Name:</td>
<td><input type="text" name="s_name" size="40"></td>
</tr>
<tr><td align="right">Telephone:</td>
<td><input type="text" name="s_telephone" size="40"></td>
</tr>
<tr><td align="right">ZIP:</td>
<td><input type="text" name="s_zip" size="40"></td>
</tr>
<tr><td align="right">email:</td>
<td><input type="text" name="s_email" size="40"></td>
</tr>
</table>
<p><input type="submit" value="Submit" name="B1"></p>
</form>
```

Name:

Telephone:

Zip code:

email:

O programa cgi para esse caso é mostrado abaixo.

```
// formdecode.cpp
#include "vbmcgi.h"
int main () {
    VBMcgi cgi;
    cgi.formDecode(true); // decode form variables and add variables
    // to the "string change feature"

    // open html file, execute string changes and put to browser
    cgi.out("person.html");
    return 0;
}
```

Veja esse exemplo na funcionando na web, na url abaixo.

http://www.vbmcgi.org/index.shtml?s_p=formDecode_adds_variables.html

14.6.6 Usando a funcionalidade “call function”

A funcionalidade “call function” (chamar função) é a segunda das funcionalidades da biblioteca VBMcgi permite o “isolamento entre o webmaster e o webdesigner” - a regra de ouro da VBMcgi, a característica que dá distinção a essa biblioteca. A primeira funcionalidade é o “string change feature”. O método “out” implementa ambas as funcionalidades (“string change” e “call function”).

A idéia da funcionalidade “call function” é permitir ao webmaster fazer programas que registram uma ou mais funções no objeto vbmcgi. As funções do usuário (usuário da VBMcgi, isto é, webmaster) necessariamente são funções globais que tem o protótipo como mostrado abaixo.

```
void userFunction(vbmcgi & userCgi, void *p);
```

Registra-se uma função global do usuário com o método (na verdade uma macro) addFunction. O método “out” irá procurar pelas funções registradas precedidas pela palavra reservada VBMCGI_CALL. Isto é, se o código html (a rigor qualquer texto no arquivo cujo nome é argumento do método “out”) contiver a seqüência de char VBMCGI_CALL(userFunction), essa seqüência não

será copiada para o console, e no seu lugar será enviado aquilo que a função `userFunction` enviar para o console. Essa funcionalidade adequa-se bem para páginas web que tem design complexo, mas contém partes interativas que são relativamente simples.

Seja o exemplo tutorial “call function feature” da página VBMcgi.org. Deseja-se fazer uma página em que um trecho do html seja na verdade fornecido por uma função do usuário (usuário aqui entende-se por “usuário de VBMcgi”, ou seja, o webmaster). Um webdesigner desenvolve com sua ferramenta de autoria html predileta a página “showMultiplyTable.html” (mostreTabuada.html). Nessa página, além de incluir uma string para ser trocada pela “string change feature” - no caso `s_mtable` -, o webdesigner inclui também a sequência `VBMCGI_CALL(multiplyTable)`. O webmaster faz um programa cgi em que existe uma função global com protótipo como mostrado abaixo, compatível com as funções do usuário que podem ser registradas no objeto `vbmcgi`.

```
void multiplyTable(vbmcgi & userCgi, void *p);
```

A função `multiplyTable` pode ler variáveis do form a partir da referência para objeto `vbmcgi` que recebeu, desde que o método `formDecode` tenha sido chamado antes de se chamar o método “out”. Para exteriorizar uma tabuada (*multiply table*), o conteúdo da variável de formulário chamada de `m_table`, e converte-se seu conteúdo para string. A partir daí, é fácil se fazer laço de programação que exteriorize a tabuada propriamente dita.

```
// call_function.cpp
#include "vbmcgi.h"
```

```
void multiplyTable(vbmcgi & userCgi, void *p) {
    VBString mt_str = userCgi.getVarContent("s_mtable"); // get as string
    int multiply_table = atoi(mt_str); // convert to int
    for (int i=1 ; i <= 10 ; i++)
        cout << i*multiply_table << "<br>" << endl;
}

int main () {
    VBMcgi cgi;
    cgi.formDecode(true); // decode form vars and add to "string change"s
    cgi.addFunction(multiplyTable);
    cgi.out("showMultiplyTable.html");
    return 0;
}
```

Seja uma página web que é um mero formulário html conectado a um programa cgi. Por clicar no botão “send”, a página html irá para `call_function.cgi`, que por sua vez tem como parâmetro o arquivo `showMultiplyTable.html`. Devido a funcionalidade “call function”, o resultado será como mostrado na figura abaixo.

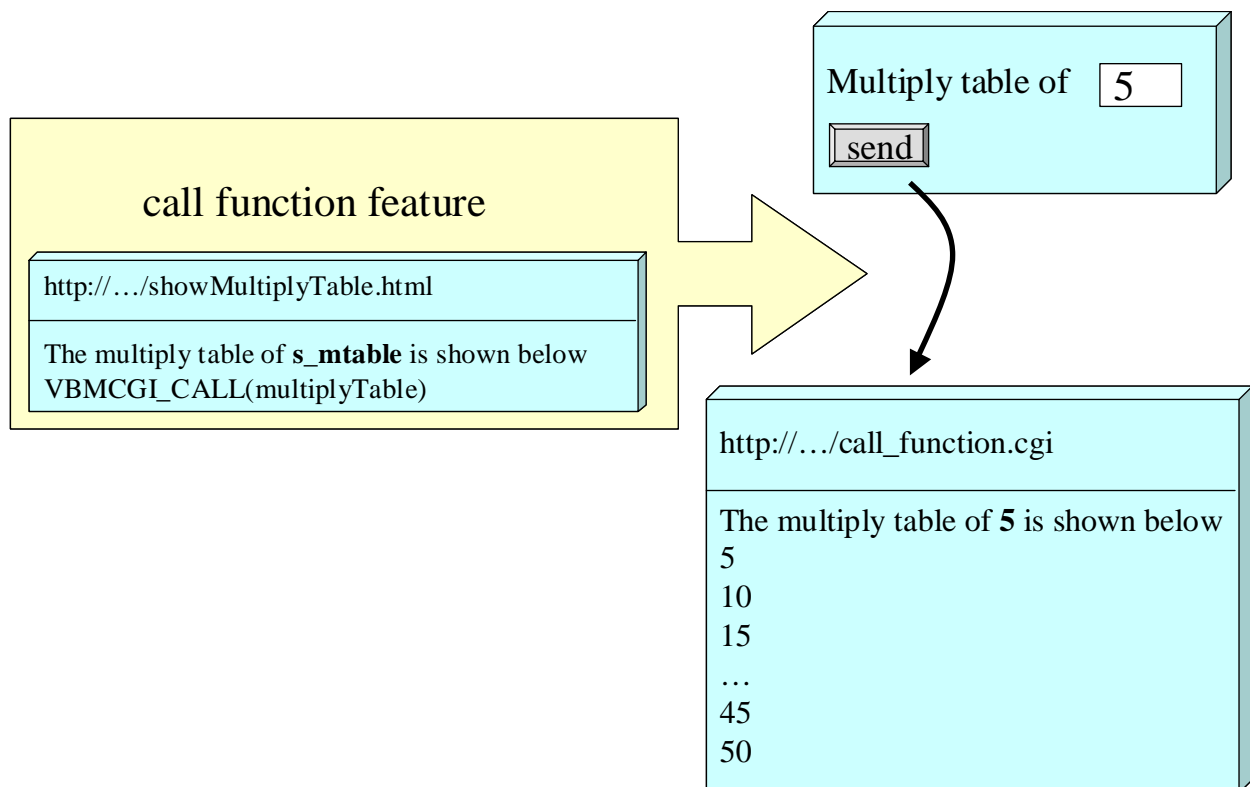


Figura 39: “call function feature”

Esse exemplo pode ser visto funcionando na web no endereço abaixo.

http://www.vbmcgi.org/index.shtml?s_p=call_function_feature.html

Outro exemplo de uso de “call function feature”: seja uma página web de uma loja virtual que trabalha com o conceito de “carro de compras”. O webdesigner pode usar a ferramenta de design html que quiser, e produzir uma página com código html sofisticado, sendo que a tabela que contém aquilo que o consumidor está comprando (o tal carro de compras) é uma tabela html no meio desse código. Em outras palavras, o que o usuário final vê é uma página complexa, com um pedaço simples (esse pedaço é a tabela com o conteúdo do carro de compras). Muito provavelmente, o conteúdo da tabela html são os campos das tabelas do banco de dados. O webmaster pode implementar essa funcionalidade com VBMcgi por produzir uma função chamada digamos “shoppingCar”, que lê os campos do carro de compras do usuário do banco de dados e joga o resultado no console como um trecho de código html. Fazer uma função como essa não é um grande desafio para o webmaster. O webdesigner inclui o banco de dados na página por colocar a sequência VBMCGI_CALL(shoppingCar) no lugar onde quer ver a tabela.

Ainda um outro exemplo: deseja-se fazer um formulário para consulta de banco de dados, digamos de imóveis. Suponha que os dados estejam classificados com alguns parâmetros, tais como bairro, número de quartos, estado da federação,

cidade, etc. Suponha que o usuário queira procurar imóveis no estado A. A partir do conhecimento do estado, sabe-se as cidades que existem. Essa informação pode estar no banco de dados. Pode-se fazer um sistema em que por se escolher o estado, a drop down do formulário que indica as cidades a se escolher seja gerada por uma user function (de call function feature). Essa user function seria algo como “pegaCidades”. O que essa função faria é enviar uma query para o banco de dados, e obter as cidades que existem num dado estado. Um laço de programa simples envia comandos html que compõe a drop down com as opções vindas do banco de dados. Portanto, o usuário final somente vê as opções de cidades que existem para o estado previamente escolhido. Continuando com a filosofia, pode-se escolher o bairro de uma lista que veio a partir do estado e da cidade. Além disso, caso exista no banco de dados apenas apartamentos com 1 e 3 quartos (sem 2 quartos) com o estado, cidade e bairro escolhidos, a opção “2 quartos” nem apareceria como opção. Se mais tarde um apartamento de 2 quartos nesse local fosse acrescentado ao banco de dados, a opção “2 quartos” apareceria para o usuário final. Tudo isso sem necessidade de se recompilar os programas cgi.

14.6.6.1 *Passando parâmetros para a função da “call function”*

Por vários motivos, pode ser necessário passar parâmetros para a função que se chama com a funcionalidade de “call function”. Um dos motivos é para permitir uso de funcionalidades do objeto vbmcgi que chama o método “out”. Uma referência do objeto vbmcgi é passada para a função do usuário. Portanto, é bastante fácil por exemplo obter o conteúdo de uma das variáveis do formulário que foi decodificado pelo objeto vbmcgi com o método formDecode (chamado apenas uma vez, fora da userFunction). Essa característica é mostrada no exemplo anterior da “call function feature”. A questão agora é como passar parâmetros extras para essa função.

Como não se pode saber antecipadamente quais são os parâmetros que se vai passar para a userFunction, a solução é ter um parâmetro adicional do tipo “void * ” (ponteiro para indeterminado). Uma variável do tipo “ponteiro para indeterminado” é um ponteiro como outro qualquer, apenas não se determinou a priori para que tipo de dados o ponteiro está apontando. Esse tipo de dados pode ser mascarado (*type cast*) para apontar qualquer tipo de dados. Portanto, para se passar qualquer parâmetro para a userFunction, basta que exista um objeto a passar como parâmetro no escopo que chama o método “out”. Para registrar a função do usuário, no caso de haver parâmetros, usa-se o método (na verdade uma macro) addFunctionObj, com dois parâmetros: a função do usuário e o objeto que se está passando como parâmetro. Dentro da função do usuário, o argumento tipo “ void * ” aponta para o objeto passado pelo método addFunctionObj. Para facilitar o uso do objeto, geralmente dentro da função do usuário copia-se o ponteiro para indeterminado para um ponteiro para o objeto verdadeiro.

No exemplo da página VBMcgi, executa-se o laço que calcula a tabuada dentro de uma função do usuário que está registrada para a funcionalidade “call function” do objeto vbmcgi. Veja o link abaixo.

http://www.vbmcgi.org/index.chtml?s_p=calling_functions_with_parameters.html

O código fonte desse exemplo está copiado abaixo. Existe uma classe do usuário `userClass`, com alguns atributos. Na função `main`, está instanciado um objeto do tipo `userClass` chamado `userObject`. Carrega-se os atributos desse objeto com alguns valores. Para se registrar a função `multiplyTable`, usou-se o método `addFunctionObj`, e passou-se o nome da função e o objeto. Dentro da função `multiplyTable`, cria-se um ponteiro `p_userClass` para a `userClass`, e com maquiagem de tipos (*type cast*), copia-se o argumento `p` para `p_userClass`. Pronto, já se tem um ponteiro válido para o objeto que foi passado como parâmetro. Um pequeno código que vem a seguir acessa os atributos do `userObject`. O operador `->` é usado para permitir que o ponteiro acesse os atributos do objeto.

```
// call_function_with_parameters.cpp
#include "vbmcgi.h"

class userClass {
public:
    int m_i;
    double m_d;
    VBString m_str;
};

void multiplyTable(vbmcgi & userCgi, void *p) {
    userClass *p_userClass; // pointer to user class
    // make type casted p point to pointer to user class
    p_userClass = (userClass*)p;

    // put some html to console,
    // using the values that were received as parameters
    cout << "<hr>" << endl;
    cout << "The value of i is " << p_userClass->m_i << "<br>" << endl;
    cout << "The value of d is " << p_userClass->m_d << "<br>" << endl;
    cout << "The value of str is " << p_userClass->m_str << endl;
    cout << "<hr>" << endl;

    // now do the standard multiply table
    // get mtable as string
    VBString mt_str = userCgi.getVarContent("s_mtable");

    int multiply_table = atoi(mt_str); // convert to int
    for (int i=1 ; i <= 10 ; i++)
        cout << i*multiply_table << "<br>" << endl;
}

int main () {
    userClass userObject; // instantiate a user object
    // fill with some data
    userObject.m_d = 2.2;
    userObject.m_i = 4;
```

```

userObject.m_str = "abc";

VBMcgi cgi;
cgi.formDecode(true); // decode form vars and add to "string change"

// register the function and the object
cgi.addFunctionObj(multiplyTable,userObject);

// call the html out method
cgi.out("showMultiplyTable.html");
return 0;
}

```

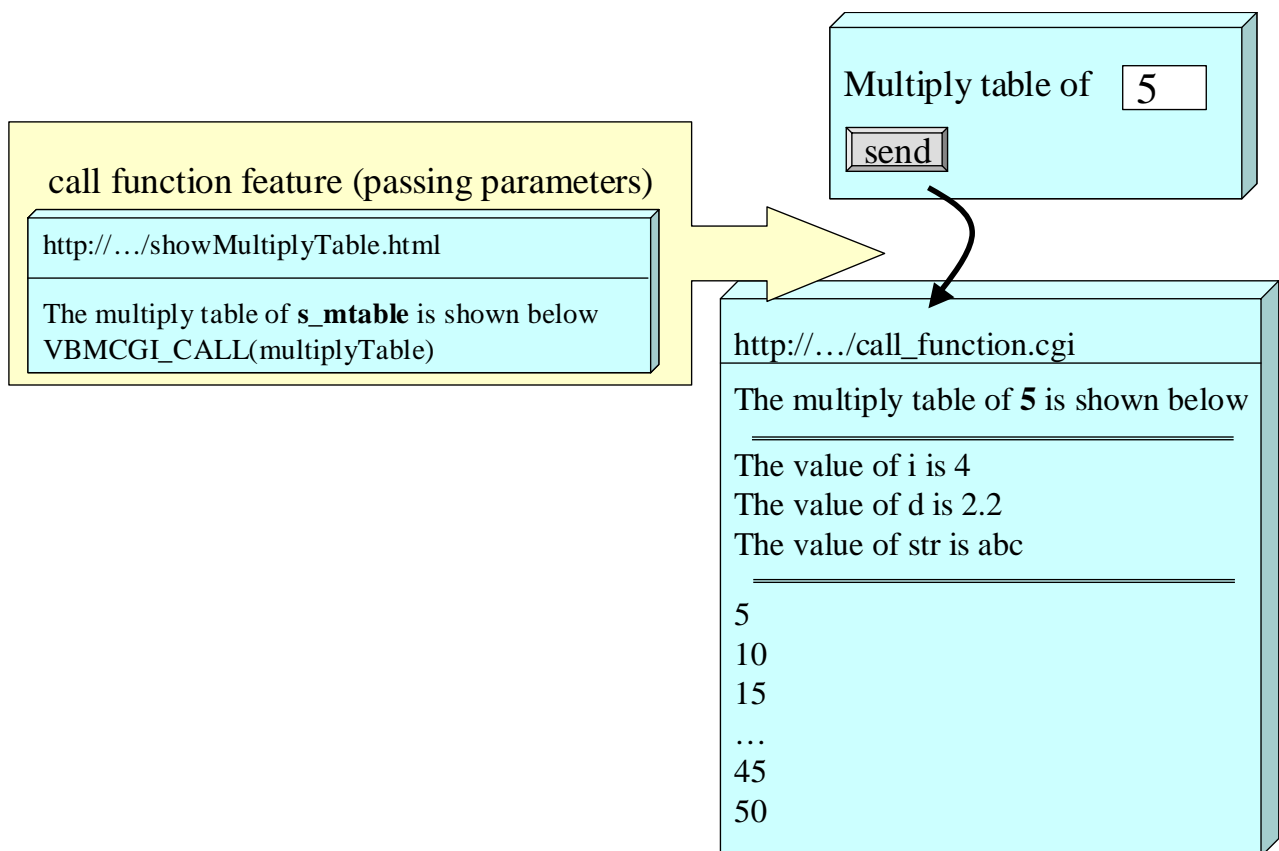


Figura 40: “call function feature”, com passagem de parâmetros para a função do usuário

Ressalte-se que como mostrado no exemplo da página VBMcgi, pode-se passar para a função do usuário um número ilimitado de parâmetros. Para fazê-lo basta que se crie uma classe do usuário, cujos atributos sejam os parâmetros que se deseja passar para a função. Essa classe pode ter quantos atributos se queira. É preciso um objeto dessa classe seja instanciado, e que seus atributos sejam carregados com os valores que se deseja passar. Por passar o ponteiro para o objeto dessa classe, se está realmente passando todos os atributos do objeto como parâmetros.

14.6.7 Redirecionando um programa cgi

Um programa cgi pode redirecionar o internauta para a url que desejar. Para isso pode usar o método “redirect” do objeto vbmcgi. Como exemplo, uma página html pode ler uma url de um formulário

```
Redirect to:
<form method="get" action="redirect.cgi">
<input type="text" name="redirect"
value="http://www.cplusplus.com/" size="80">
<input type="submit" value="Submit">
```

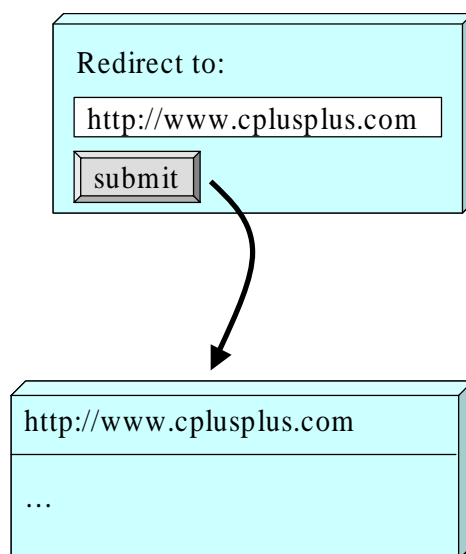


Figura 41: redirecionamento de url em um programa cgi

O código fonte do programa redirect.cgi é mostrado abaixo.

```
// redirect.cpp
#include "vbmcgi.h"
int main () {
    VBMcgi cgi;
    cgi.formDecode();
    VBString url = cgi.getVarContent("redirect");
    cgi.redirect(url);
    return 0;
}
```

A saída do programa é como mostrado abaixo.

```
status: 301
location: http://www.cplusplus.com/
```

Um exemplo de redirecionamento de url por programa cgi é mostrado na url abaixo.

http://www.vbmcgi.org/index.chtml?s_p=redirection.html

14.6.8 Cookies

Os cookies são um recurso importante da programação web, relativamente pouco compreendido. Há muito o que falar somente sobre cookies. Nessa seção se discutirá muito resumidamente o que são os cookies, focalizando-se em como usa-los em programas cgi usando C++ e VBMcgi. Veja informações adicionais sobre cookies em “Cookie Central” [64], ou em [41].

O cookie é uma informação que o servidor deposita no cliente para poder identifica-lo mais tarde. Essa informação é um par de strings - `cookieName` e `cookieValue` - com alguns parâmetros adicionais tais como data de expiração. A partir do uso de cookies, pode-se definir o conceito de “estado” num sistema na web. Sem o cookie, cada acesso que o cliente requer ao servidor seria indiferenciável. Com o cookie, pode-se diferenciar o acesso. Um exemplo concreto e bem útil é o caso em que o usuário “faz login” (entra) num sistema desenvolvido para web. Esse é o caso do Internet Bank. Para alguém “fazer login” (entrar) num sistema, claramente existe o estado “dentro” (logado) e o estado “fora” (não logado). A identificação de um e outro estado se dá pela existência ou não-existência de cookie.

Um profissional criativo pode imaginar várias formas de se aproveitar o conceito de estado para um sistema de software na web. Pode-se ter estado para “usuário cadastrado” e “usuário não cadastrado”. Para o usuário cadastrado, que “loga” na sua página e tem senha conferida, oferece-se um tipo de conteúdo. Para usuário não cadastrado, oferece-se outro conteúdo. Oferecer conteúdo customizado (adaptado para cada cliente) é outra capacidade derivada do uso de cookies. Cada link de um sistema na web pode estar ligado a programas cgi que checam, baseados em cookies, informações sobre o internauta, e com isso mostra-se o conteúdo de forma apropriada.

Em concordância com a especificação do protocolo http, o cookie fisicamente é enviado dentro do cabeçalho html. Um programa cgi pode enviar um cookie por enviar cookies simplesmente por gerar uma string que é o próprio cookie e enviar essa string no cabeçalho, isto é, antes de “fechar” o cabeçalho. O cabeçalho termina com o envio de duas letras `<return>` seguidas. Pode-se enviar zero, um ou mais de um cookie num mesmo cabeçalho.

O exemplo abaixo mostra como um programa qualquer enviaria 3 cookies para o navegador do cliente. Repare uma linha em branco entre o `cookie_3` e o início do código html. Essa linha em branco, considerando o `<return>` da linha anterior, completa a seqüência de 2 `<returns>`, o que caracteriza o fim do html cabeçalho.

```
Content-type:text/html
<cookie_1>
<cookie_2>
<cookie_3>

<html>
...
```

</html>

O programa cookies_3.cpp abaixo envia 3 cookies constantes para o cliente.

```
// cookies_3.cpp
#include "vbmcgi.h"

int main()
{
    vbmcgi cgi;
    cgi.htmlHeader(); // begin the html header

    // set parameters of cookie 1
    cgi.setCookieNameValue("cookie name 1", "cookie value 1");
    cgi.setCookieExpires("end-of-session");
    cgi.sendCookie(); // send cookie to browser

    // set parameters of cookie 2
    cgi.setCookieNameValue("cookie name 2", "cookie value 2");
    cgi.setCookieExpires("end-of-session");
    cgi.sendCookie(); // send cookie to browser

    // set parameters of cookie 3
    cgi.setCookieNameValue("cookie name 3", "cookie value 3");
    cgi.setCookieExpires("end-of-session");
    cgi.sendCookie(); // send cookie to browser

    cout << endl; // end the html header

    cgi.out("test.html"); // html code to the console
    return 0;
}
```

A saída do programa é como mostrado abaixo.

```
Content-type:text/html
Set-Cookie: cookie name 1=cookie value 1;expires=
Set-Cookie: cookie name 2=cookie value 2;expires=
Set-Cookie: cookie name 3=cookie value 3;expires=

<html>
test.html
</html>
```

14.6.8.1 Configuração do navegador para ver os cookies

Na maioria das vezes, o navegador do usuário implicitamente aceita os cookies sem dar nenhum alerta. Mas pode-se reconfigurar o navegador para que o cookie gere uma caixa de diálogo, perguntando se o internauta aceita ou não, antes de realmente aceitar o cookie.

Veja como configurar seu navegador para ver cookies na url abaixo.

<http://www.vbmcgi.org/youReceivedACookie.html>

14.6.8.2 Expiração

Um parâmetro dos cookies é a sua data-hora de expiração. Com respeito a sua expiração os cookies são de dois tipos.

- Cookie tipo “end-of-session” (fim da sessão), também chamado de “per-session” (por sessão).
- Cookies persistente, ou cookie em arquivo.

O cookie tipo “end-of-session” tem data-hora de expiração marcada pela string “end-of-session”. Se estiver habilitado para aceita-los, o navegador irá armazenar a informação do cookie na memória. Portanto, caso se desligue o computador, qualquer informação desse tipo de cookie se perde. Esse tipo de cookie é muito usado para sistemas que logam o internauta para uma finalidade específica. Por exemplo: Internet Bank. Nesse caso, a finalidade do cookie é checar se o usuário passou pela conferência de senha ou não, e portanto não há necessidade de se armazenar essa informação depois que o usuário desliga o computador.

O cookie em arquivo tem uma data-hora de expiração explicitamente definida, e faz aparecer um arquivo no computador do internauta (se este permitir o uso de cookies). Enquanto a data-hora de expiração não for ultrapassada, o arquivo de cookie é válido. Após essa data-hora, o cookie é ignorado e o seu arquivo apagado. Esse tipo de cookie é muito usado para produzir conteúdo customizado (isto é adequado para cada usuário). Por exemplo: seja um internauta que acessa a web sempre do mesmo computador. Digamos que ele comprou um livro na amazon.com sobre o assunto C++. O sistema da amazon.com manda para o usuário um cookie em arquivo, contendo um número que identifica esse usuário no seu banco de dados. Num outro dia em que o usuário entre no site da amazon.com, aparecerão sugestões de livros de C++ para comprar (característica clara de conteúdo customizado). Como isso é feito ? O sistema web da amazon, todo feito baseado em programas cgi, verifica se o internauta tem um cookie da amazon. Se tiver, consulta no banco de dados as preferências desse cliente e baseado num algoritmo de sugestões de vendas produz um conteúdo customizado para esse cliente. Se não tiver um cookie da amazon, assume-se que esse é um cliente novo e então envia-se um novo número para esse cliente, e além disso executa-se um procedimento de exibição de conteúdo padrão.

O diretório do computador do internauta em que se armazena os arquivos que são os cookies depende do sistema operacional e do navegador. Se esses arquivos forem apagados, toda a informação dos cookies em arquivo se perde. Ou seja, da próxima vez que se comprar um livro na amazon.com, não surgirá sugestões de livros ao gosto do internauta. Para que se apague os cookies “per-session”, basta fechar o aplicativo que é o navegador e inicia-lo novamente.

Quando um site pede para que o internauta vote em algo, e o impede de votar mais de uma vez por dia, essa informação é armazenada em cookies. Por apagar os cookies, pode-se votar quantas vezes quiser.

O formato de data-hora para cookies persistente é como mostrado no exemplo abaixo.

```
// persistent_cookie.cpp
#include "vbmcgi.h"
int main ()
{
    VBMcgi cgi;
    cgi.htmlHeader(); // begin the html header

    // set parameters of cookie
    cgi.setCookieNameValue("cookie name","cookie value");
    cgi.setCookieExpires("31-Mar-2002 23:59:59 GMT");
    cgi.sendCookie(); // send cookie to browser

    cout << endl; // end the html header

    cgi.out("test.html"); // html code to the console
    return 0;
}
```

A saída do programa é como mostrado abaixo.

```
Content-type:text/html
Set-Cookie: cookie name=cookie value;expires=31-Mar-2002 23:59:59 GMT

<html>
test.html
</html>
```

14.6.8.3 Domínio de um cookie

Um cookie transporta a informação do domínio que o gerou. Portanto, se o site `dominio_A.com` envia para o internauta o cookie com `cookieName` contendo “userId” (não importa o `cookieValue`), e se o site `dominio_B.com` manda um outro cookie o cookie com `cookieName` contendo “userId”, esses dois cookies vão co-existir no computador do internauta, sem que um cookie sequer saiba da existência do outro cookie.

Pode-se fazer uma página na url `dominio_A.com`, que tem uma figura (pode ser um banner), com link, hospedado em `dominio_B.com`. Se o internauta clica na figura em questão, e se essa figura enviar um cookie para o internauta, o domínio do cookie será `dominio_B.com`, porque é nesse domínio que o internauta está de fato clicando e recebendo informações. Mas ressalte-se que isso ocorre enquanto o internauta observa `dominio_A.com` no campo url do seu navegador.

14.6.8.4 Sistema na web com login usando cookies

Seja o exemplo abaixo de sistema na web que permita login, usando cookies. Nesse exemplo, há 2 programas cgi, e 5 arquivos html. A figura abaixo ilustra o diagrama de estados do sistema. Nesse sistema, há 3 procedimentos “protegidos”, que somente podem ser executados por quem teve conta e senha conferidas. Há uma página inicial `login.html`. Essa página tem um formulário html que conduz ao programa `login.cgi`, que confere a conta e senha (no caso do exemplo, a conferência é tipo “hard coded”, mas poderia ser baseada em banco

de dados ou outra forma qualquer). No caso de a senha não conferir, o login.cgi exterioriza a página `please_login.html` que informa o fato de a senha estar incorreta, e leva o internauta de volta a `login.html`. No caso de a senha estar certa, o login.cgi envia um cookie para o internauta e exterioriza a página `procedure_menu.html`, que oferece as opções de procedimentos protegidos, e também um procedimento de logout.

O cookie enviado, nesse caso, é um cookie constante, definido de forma “hard coded”. Num caso mais real, login.cgi seria melhor gerar um cookieValue com um número aleatório, e salvar esse número num arquivo local no servidor. No exemplo tutorial, o que identifica o estado “logado” ou não é a existência ou não do cookie com cookieValue correto.

Os procedimentos protegidos são executados necessariamente por executar o programa `logged.cgi`. No exemplo tutorial, o procedimento protegido foi reduzido a exibição dos arquivos `procedure_n.html`. Essa exibição nesse caso é um “out” do arquivo original em html. Acontece que o programa `logged.cgi` sempre confere a existência do cookie antes de executar os procedimentos protegidos. Portanto, caso um hacker tente executar esse programa diretamente sem antes ter passado pelo procedimento de conferência de senha (e nesse caso não teria recebido o cookie), o `logged.cgi` não executa o procedimento protegido, mas exterioriza o `cannot_run_pocedure.html` (`não_pode_executar_procedimento.html`), que por sua vez encaminha o internauta para a página de login. O programa `logged.cgi` lê o cookie com o método `getCookieValue`.

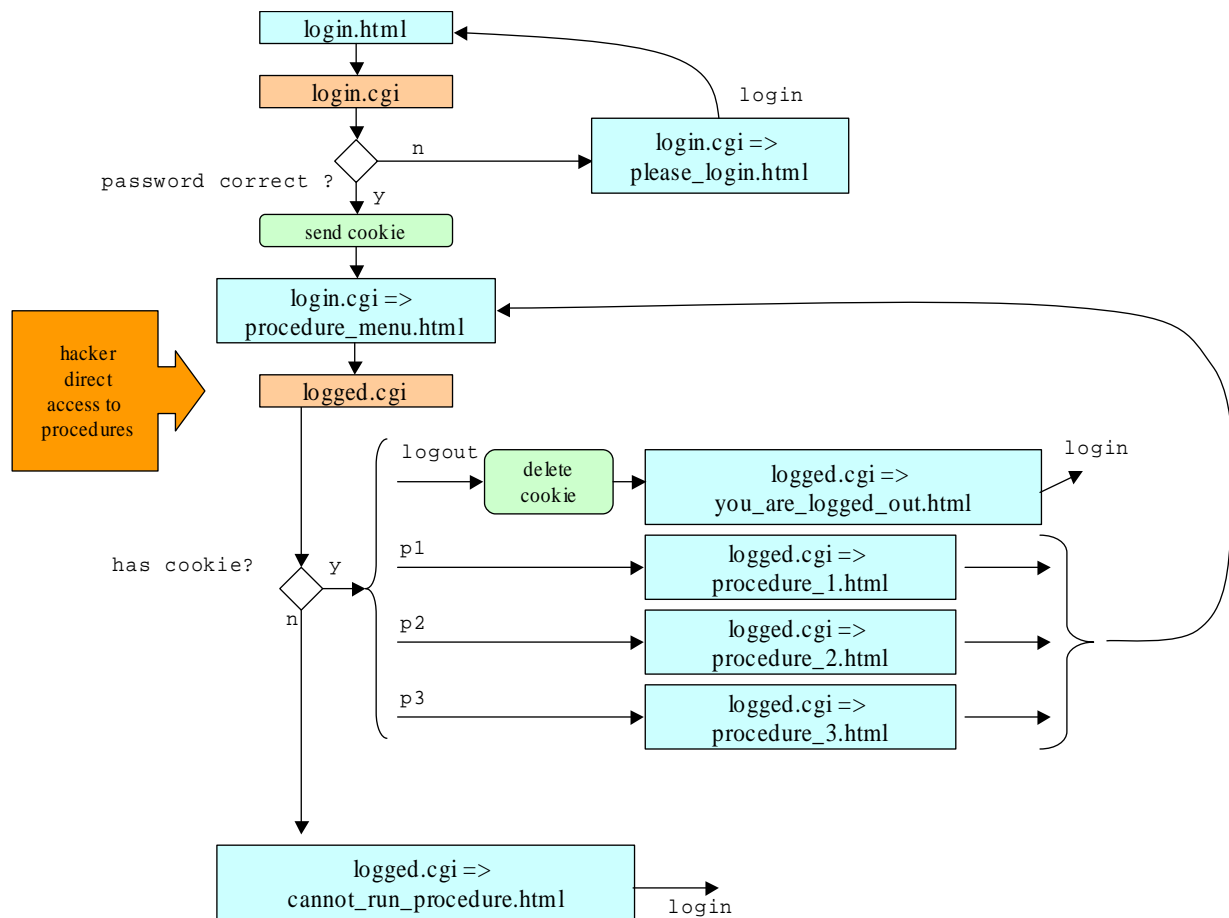


Figura 42: mapa de um sistema na web com login, usando cookies

Veja o exemplo tutorial em execução na url abaixo.

http://www.vbmcgi.org/index.shtml?s_p=login_using_cookies.html

O código fonte do programa cgi login.cpp é mostrado abaixo.

```

// login.cpp
#include "vbmcgi.h"

int main () {
    VBString correctLogin = "abc";
    VBString correctPasswd = "def";
    VBString cookieName = "user_id";
    VBString cookieValue = "1234";

    vbmcgi cgi;
    cgi.formDecode();
    VBString login = cgi.getVarContent("s_login");
    VBString passwd = cgi.getVarContent("s_passwd");
    bool loginOK = ((correctLogin == login) &&
        (correctPasswd == passwd));

    if (loginOK) {
        cgi.htmlHeader();
        cgi.setCookieNameValue(cookieName,cookieValue);
    }
}
  
```

```

        cgi.setCookieExpires("end-of-session");
        cgi.sendCookie();
        cout << endl; // close header
        cgi.out("procedure_menu.html");
    }
    else { // if login is not OK
        cgi.out("please_login.html");
    }
    return 0;
}

```

O código fonte do programa cgi logged.cpp (logado) é mostrado abaixo.

// logged.cpp

```

#include "vbmcgi.h"

int main () {
    VBString cookieName = "user_id";
    VBString correctCookieValue = "1234";

    vbmcgi cgi;
    VBString cookieValue = cgi.getCookieValue(cookieName);
    if (cookieValue != correctCookieValue) {
        cgi.out("cannot_run_procedure.html");
        exit(1);
    }
    cgi.formDecode();
    VBString procedureStr = cgi.getVarContent("s_proc");
    int proc = atoi(procedureStr);

    switch (proc) {
    case 0: // logout
        cgi.htmlHeader();
        cgi.setCookieNameValue(cookieName,""); // send a blank cookie to logout
        cgi.setCookieExpires("end-of-session");
        cgi.sendCookie();
        cout << endl; // end of header
        cgi.out("you_are_logged_out.html");
        break;
    case 1: // procedure 1
        cgi.out("proc_1.html");
        break;
    case 2: // procedure 2
        cgi.out("proc_2.html");
        break;
    case 3: // procedure 3
        cgi.out("proc_3.html");
        break;
    case 4: // procedure menu
        cgi.out("procedure_menu.html");
        break;
    default:
        cgi.out("please_login.html");
        break;
    } // end of switch;
    return 0;
}

```

14.6.9 Programas cgi com dados não html

Um programa cgi não precisa retornar necessariamente dados html. Pode-se retornar vários tipos de dados. Por exemplo, dados de arquivo gráfico (binário) tipo gif. Para isso, basta enviar um cabeçalho para o tipo de dado em questão, e em seguida os dados. No exemplo abaixo, envia-se um cabeçalho para imagem do tipo gif, e em seguida envia-se binariamente, byte a byte, o conteúdo de um arquivo gráfico do tipo gif. Com isso, o resultado é um programa cgi que se comporta como uma figura gif.

http://www.vbmcgi.org/index.shtml?s_p=gif_output.html

```
// gif_output.cpp

#include <fstream.h>
#include "vbmcgi.h"

#ifdef WIN32
    #define BIN ,ios::binary
#else
    #define BIN
#endif

int main() {
    VBMcgi cgi;

    const char *file = "vbmcgi_logo.gif";

    ifstream gifFile(file BIN);
    if (!gifFile) {
        cgi.htmlCompleteHeader();
        cout << "Could not open gif file for reading" << endl;
        exit(1);
    }

    cgi.htmlCompleteHeader("image/gif");
    unsigned char z;
    while (true) {
        gifFile.read(&z, sizeof(char));
        if (gifFile.eof()) break;
        cout.write(&z, sizeof(char));
    }
    return 0;
}
```

A saída desse programa é como mostrado abaixo.

Content-type:image/gif

(letras aleatórias, que correspondem a informação binária do arquivo gif convertidas para texto. No navegador do internauta, aparece tudo ok.)

14.7 Contador de página

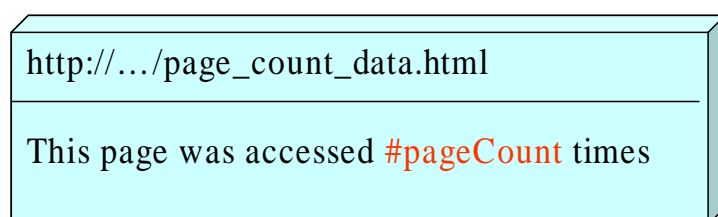
Um recurso clássico de programas cgi é o contador de página. Esse recurso indica quantas vezes uma página foi acessada. Por ativar a opção “reload” (recarregar) do navegador do internauta, o contador indica o incremento no número de vezes que a página foi acessada. Na biblioteca VBMcgi há uma classe

VBPageCount, específica para essa finalidade. O construtor dessa classe tem o protótipo como mostrado abaixo.

```
VBPageCount::VBPageCount(const char *id, const char*fileName="VBPageCount.txt");
```

O primeiro parâmetro é o identificador da contagem, e o segundo - implícito - é o nome do arquivo onde se vai armazenar as contagens. Implicitamente o nome desse arquivo é VBPageCount.txt.

Seja uma página html de molde para apresentação do número de vezes que foi acessada. A figura abaixo ilustra uma página simples para esse tipo de apresentação.

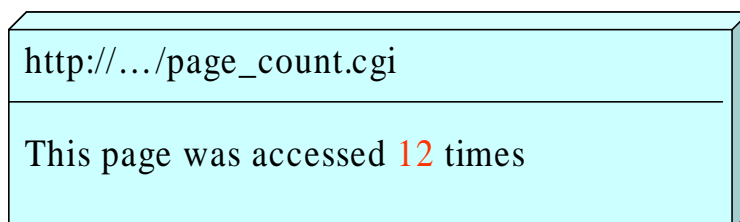


Por usar a classe VBPageCount, o programa abaixo cuida de armazenar em arquivo a informação de quantas vezes a página foi acessada. Por usar a “string change feature”, pode-se usar a página de molde acima para efetivamente mostrar o número de vezes que a página foi acessada. A string de procura no caso é “#pageCount”, e a string de troca é aquela que retorna do método getCount, da classe VBPageCount.

```
// page_count.cpp
#include "vbmcgi.h"
int main () {
    vbmcgi cgi;
    VBPageCount myPageCount("page_count");
    cgi.addBySource("#pageCount",myPageCount.getCount());
    cgi.out("page_count_data.html");
    return 0;
}
```

O efeito final é como mostrado na figura abaixo. Por recarregar a página html, o número é incrementado. Veja também o exemplo no link abaixo.

http://www.vbmcgi.org/index.shtml?s_p=page_count.html



Uma observação importante é que esse exemplo requer direitos de escrever em disco para programas cgi. Em unix, por exemplo, dependendo da configuração do servidor web, os programas cgi são executados com identificação de

“nobody”, ou do próprio usuário (no caso de a opção suexec estar instalada). O usuário “nobody” não tem em princípio direito de escrever no diretório do usuário em questão. Uma solução é dar permissão a todo o mundo de escrever num dado diretório. Mas essa solução gera problemas de segurança. Em geral é melhor instalar a opção “suexec” do apache, e com isso o usuário executa programas cgi com sua própria identificação.

Caso se esteja usando hospedagem virtual (*virtual host*) – recurso que permite o apache a hospedar mais de um domínio no mesmo computador, com o mesmo número IP, a configuração da hospedagem virtual deverá definir que os programas cgi do domínio virtual em questão são executados com a identificação de um determinado usuário. Esse usuário deverá ter direitos de escrita sobre o diretório em que se pretende armazenar o arquivo onde são salvas as informações de acesso em página.

Veja mais detalhes de configuração do servidor web apache em www.apache.org.

14.8 Calendário

Por muitos motivos, pode-se precisar de um calendário num sistema na web. O cálculo correto do dia da semana de um dado mês e ano costuma ser uma função que não se tem, e que se demora a desenvolver. Na VBMcgi há uma classe chamada VBTableMonth, que ajuda bastante para a elaboração de um calendário na web. Instancia-se um objeto da classe. Com o método setDate, define-se mês e ano que se quer do calendário. Adicionalmente, pode-se acrescentar qualquer conteúdo (código html) dentro de cada dia do calendário, usando o método setDayContent. O método htmlOut envia para o console o código que serve de tabela html para que apareça um calendário no navegador do cliente final.

```
#include "vbmcgi.h"

const char *comment="comment";

void myCalendar(vbmcgi & cgi, void *p) {
    VBTableMonth a;
    int year = 2002;
    int month = VBmar; // VBjan = 0, VBfeb = 1, VBmar = 2, ...
    a.setDate(year,month);
    a.setDayContent(31,comment);
    a.htmlOut();
}

int main () {
    VBMcgi cgi;
    cgi.addFunction(myCalendar);
    cgi.out("table_month_data.html");
    return 0;
}
```


Pode-se chamar o método `setDayContent` mais de uma vez, até uma vez para cada dia do mês. Se o dia do mês não existir, o comando será ignorado. No exemplo, mostra-se Março de 2002, com um comentário no dia 31.

Veja o exemplo relacionado na página VBMcgi.

http://www.vbmcgi.org/index.chnl?s_p=table_month.html

Nesse outro exemplo, gera-se um calendário no mês e ano que o usuário entra pelo formulário. Além disso, acrescenta-se um comentário no dia escolhido.

http://www.vbmcgi.org/index.chnl?s_p=table_month_2.html

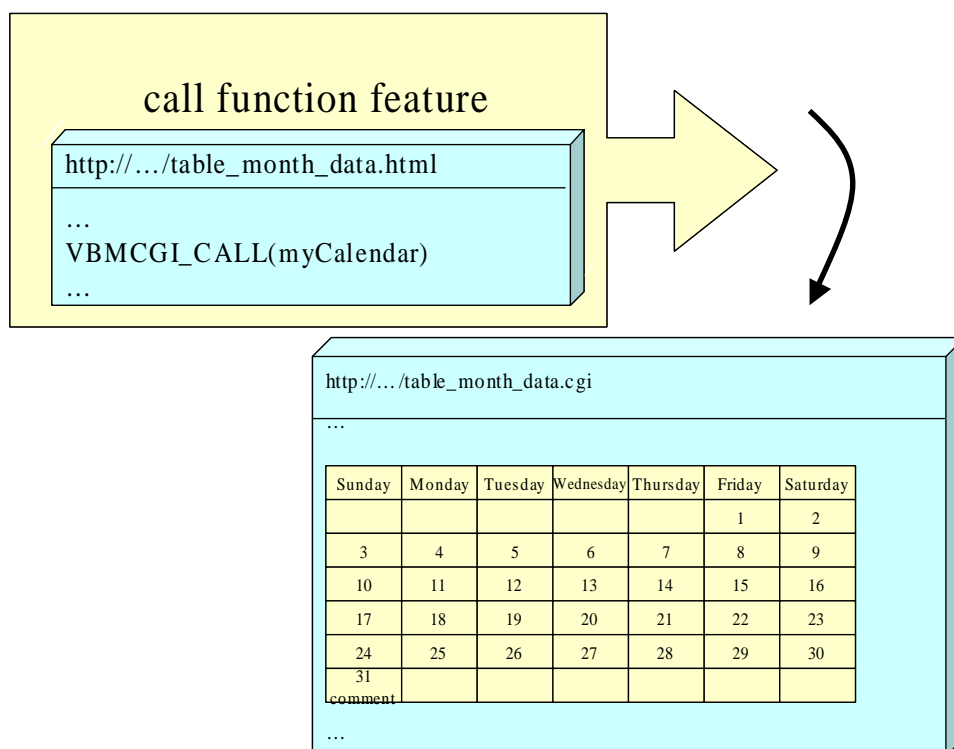


Figure 43: Calendário gerado pela classe `VBTableMonth`, usando `call function feature`

Caso o design da página html não seja de agrado do leitor, em princípio pode-se desenvolver uma classe semelhante a `VBTableMonth`, que exteriorize uma tabela com outro design.

Alguns métodos foram acrescentados a essa classe para torna-la mais flexível. Um deles é o que define os nomes dos dias de semana. Esse método é útil para que se possa usar outro idioma diferente de Inglês. O método `setWeekDays`, com protótipo abaixo, recebe os nomes dos dias de semana que serão usados pelo método `htmlOut`.

```
void VBTableMonth::setWeekDays(
```

```

const char *Sunday,
const char *Monday,
const char *Tuesday,
const char *Wednesday,
const char *Thursday,
const char *Friday,
const char *Saturday);

```

Portanto, por acrescentar a linha abaixo antes de se chamar o método `htmlOut`, o objeto `VBTableMonth` passa a gerar um calendário com nomes de dia de semana em Português.

```

a.setWeekDays("domingo","segunda-feira","terça-feira","quarta-feira",
"quinta-feira","sexta-feira","sábado");

```

14.9 Menu em arquivo

Por várias razões, pode ser muito útil que um menu seja gerado a partir de um conteúdo de um arquivo de texto externo ao programa cgi propriamente dito. Dessa forma, pode-se fazer um site na web, com diversas páginas, sendo que todas tem o mesmo menu. Por mudar o conteúdo do arquivo que descreve o menu, se está mudando o menu de todas as páginas do site. Mas para isso, é preciso que o site seja navegado integralmente por programas cgi, e não por páginas html.

O próprio site `vbmcgi.org` é um exemplo de site na web desenvolvido dessa forma. Toda a navegação desse site é feita com programas cgi. Os menus do lado esquerdo são gerados a partir de funções que geram código html, e que são incluídas na visualização do navegador do internauta utilizando-se a “call function feature”. No site `vbmcgi.org`, a navegação é feita pelo programa `index.shtml` (poderia ser `index.cgi`), que é o resultado da compilação do arquivo `index.cpp`, mostrado abaixo.

Há 3 funções acrescentadas ao “call function feature”, chamadas “menu”, “cpplib”, e “menuTutorial”. O que cada uma dessas funções faz é gerar o código do menu, sendo que cada opção está com a sua respectiva ligação html. Para efetuar o trabalho de gerar o trecho de código html que corresponde ao menu, existe a classe `VBMenuHtml`. Dentro das funções chamadas pela “call function feature”, instancia-se um objeto dessa classe, e com o método `menuFromFile`, gera-se uma saída para o console com o menu html. O parâmetro do método `menuFromFile` é um arquivo texto que descreve o menu html a ser gerado, descrito num formato próprio com extensão `*.vhm` (Villas Html Menu).

```

// index.cpp
#include <fstream.h>
#include "vbmcgi.h"

void menu(vbmcgi & cgi, void *p) {
    VBMenuHtml m;
    m.menuFromFile("mainmenu.vhm");
}

void cpplib(vbmcgi & cgi, void *p) {
    VBMenuHtml m;

```

```

        m.menuFromFile("cpplib.vhm");
    }

void menuTutorial(vbmcgi & cgi, void *p) {
    VBMenuHtml m;
    m.menuFromFile("tutorial.vhm");
    cout << "<p>" << endl;
    m.menuFromFile("tutorial_advanced.vhm");
}

int main () {
    VBMcgi cgi;
    cgi.addFunction(menu);
    cgi.addFunction(menuTutorial);
    cgi.addFunction(cpplib);
    cgi.formDecode();
    VBString link = cgi.getVarContent("s_p");
    if (link == VBString("")) { // if http://www.vbmcgi.org/
        link = "index_original.html";
        VBPageCount myPageCount("index");
        cgi.addBySource("s_indexCount",myPageCount.getCount());
    }
    cgi.out(link);
    return 0;
}

```

O formato *.vhm é um arquivo de texto que descreve um menu em html. A primeira linha desse arquivo de texto necessariamente é um comentário, sendo que a primeira letra (a letra mais a esquerda) é a letra que será usada para definir o que é o que não é comentário. Nos exemplos abaixo, a letra escolhida é o “%” (porcento). As linhas com comentário são ignoradas. A primeira linha não comentada contém a letra que será usada como separador. Nos exemplos abaixo, a letra usada é a vírgula. A próxima linha não comentada define o estilo do menu html, com parâmetros separados pela letra de separação (vírgula no exemplo). O primeiro parâmetro de estilo é o tipo de menu (por enquanto somente se pode selecionar 0 ou 1, mas no futuro haverá mais estilos). A segundo parâmetro é a cor da letra. Em seguida, vem o tamanho, ser negrito ou não, estar entre chaves ou não, ser horizontal ou não. A forma mais fácil de se entender o significado desses parâmetros é testa-lo, vendo os resultados diretamente num navegador. Opcionalmente, pode-se ver o código fonte da classe VBHtmlMenu, e adapta-la, ou desenvolver uma classe semelhante para que se gere um menu html com estilo diferente.

Prosseguindo na descrição do formato vhm, a próxima linha não comentada é o título do menu, que é uma linha de código html contendo qualquer conteúdo.

As próximas linhas não comentadas são as strings que aparecem no menu, e a sua ligação html, separados pela letra de separação (vírgula no exemplo). O conteúdo da ligação html pode ser uma url completa (por exemplo <http://www.vbmcgi.org/vblib/>), ou uma url relativa. Para manter o site vbmcgi.org todo em cgi's (e portanto evitando-se o uso de html diretamente), o conteúdo das ligações html são geralmente `index.shtml?s_p=<alguma_página>.html`. Dessa forma, se está passando diretamente pela url um parâmetro para a

query_string. No caso, a variável s_p está recebendo "<alguma_pagina>.html". Esse parâmetro é recuperado pelo programa index.shtml, e usado como argumento do método out. No caso da url <http://www.vbmcgi.org/>, o programa index.shtml é executado sem nenhuma query_string. Nesse caso o conteúdo da variável s_p é uma string vazia. O programa index.shtml nesse caso chama o método out com o parâmetro "index_original.shtml". Como curiosidade, aponte o navegador para http://www.vbmcgi.org/index_original.shtml, ou para qualquer outra página html diretamente e veja-as sem o processamento do programa cgi.

Por modificar o conteúdo das páginas html propriamente ditas, se estará modificando do que se vê no site vbmcgi.org. A ferramenta de autoria que se usa para gera as páginas html é irrelevante para o funcionamento do site. Essa característica mantém a regra de ouro da VBMcgi, que é isolar o webmaster do webdesigner.

Pelo fato de se criar um site todo usando programas cgi (sem referenciar diretamente páginas html), e se usar menu criado por arquivo, pode-se mudar os menus que o internauta observa apenas por se mudar o conteúdo da descrição do menu, no arquivo *.vhm. Por fazer-lo, se estará mudando o menu de todas as páginas do site que chamam esse menu. Dessa forma, a manutenção da navegabilidade do site é feita de forma fácil, apenas por se usar programação cgi, ou seja, sem requerer ao administrador da rede configurações especiais do servidor web.

Listagem de mainmenu.vhm é mostrada abaixo.

```
% mainmenu.vhm
% *.vhm stands for Villas-Boas Html Menu file
% the first uncommented line is the char to be used as token
,
% the second uncommented line is the meny style,
% separated by tokens of the style parameters
% style,color,size,bold,braket,horizontal
1,#ff0000,+4,true,true,false
% title of menu
<b><font color="#000066" size="+1">Main menu:</font></b><br>
% from now on these are the the menu<tok>link lines
Home,/
Download & tutorial,index.shtml?s_p=down_tut.html
History,index.shtml?s_p=history.html
Email list,index.shtml?s_p=email_list.html
Credits,index.shtml?s_p=credits.html
Strategy,index.shtml?s_p=strategy.html
Golden Rule,index.shtml?s_p=golden_rule.html
```

Listagem de cpplib.vhm é mostrada abaixo.

```
% cpplib.vhm
% *.vhm stands for Villas-Boas Html Menu file
% the first uncommented line is the char to be used as token
,
% the second uncommented line is the meny style,
% separated by tokens of the style parameters
% style,color,size,bold,braket,horizontal
1,#0000aa,+2,false,true,false
```

```
% title of menu
<font color="#000066" size="+1">C++ libraries:</font><br>
% from now on these are the the menu<tok>link lines
VBMcgi,http://www.vbmcgi.org/
VBLib,http://www.vbmcgi.org/vbllib/
VBMath,http://www.vbmcgi.org/vbmath/
VBCSsql,http://www.vbmcgi.org/vbcssql/
```

Listagem de tutorial.vhm é mostrada abaixo.

```
% tutorial.vhm
% *.vhm stands for Villas-Boas Html Menu file
% the first uncommented line is the char to be used as token
,
% the second uncommented line is the meny style,
% separated by tokens of the style paramenters
% style,color,size,bold,braket,horizontal
% 1,#0000aa,+2,false,true,false
0,#0000aa,,false,true,false
% title of menu
<font color="#000066" size="+1">Basic tutorial:</font><br>
% from now on these are the the menu<tok>link lines
Hello CGI,index.chnl?s_p=hello_cgi.html
Multiply table,index.chnl?s_p=multiply_table.html
Environment,index.chnl?s_p=environment.html
Hello VBMcgi,index.chnl?s_p=hello_vbmcgi.html
Multiply table with parameter,index.chnl?s_p=multiply_table_with_parameter.html
String parameters,index.chnl?s_p=string_parameters.html
"string change feature",index.chnl?s_p=string_change_feature.html
formDecode adds variables,index.chnl?s_p=formDecode_adds_variables.html
Complete form reference,index.chnl?s_p=complete_form_reference.html
"call function feature",index.chnl?s_p=call_function_feature.html
Calling functions with parameters,index.chnl?s_p=calling_functions_with_parameters.html
Cookies,index.chnl?s_p=cookies.html
Login to web using cookies,index.chnl?s_p=login_using_cookies.html
gif or jpeg output,index.chnl?s_p=gif_output.html
```

Listagem de tutorial_advanced.vhm é mostrada abaixo.

```
% tutorial_advanced.vhm
% *.vhm stands for Villas-Boas Html Menu file
% the first uncommented line is the char to be used as token
,
% the second uncommented line is the meny style,
% separated by tokens of the style paramenters
% style,color,size,bold,braket,horizontal
% 1,#0000aa,+2,false,true,false
0,#0000aa,,false,true,false
% title of menu
<font color="#000066" size="+1">Advanced tutorial:</font><br>
% from now on these are the the menu<tok>link lines
getDateTime,index.chnl?s_p=getDateTime.html
redirection,index.chnl?s_p=redirection.html
page count,index.chnl?s_p=page_count.html
table month,index.chnl?s_p=table_month.html
table month 2,index.chnl?s_p=table_month_2.html
```

14.10 Programação em 3 camadas e sistemas na web com VBMcgi

Muitos autores recomendam a programação de sistemas de informação - particularmente os sistemas na web - em 3 camadas de software. Resumidamente, as 3 camadas são as seguintes:

1. Camada de apresentação.
2. Camada de “regras de negócio”.
3. Camada de banco de dados.

Idealmente, os problemas de uma camada não devem afetar outra camada. No caso de sistemas na web, a camada de apresentação é o html e o design que as páginas tem. Usando VBMcgi, essa camada é basicamente feita pelo webdesigner, e eventualmente feita em pequenas quantidades pelo webmaster, quando escreve funções para o “call function feature”, que jogam código html no console.

A camada de “regras de negócio” em princípio deve ser feita pelo desenvolvimento de classes que em tese se poderia reaproveitar num outro contexto. Essas classes implementam o modelo do sistema que se está desenvolvendo. Usando VBMcgi, essa camada é basicamente feita pelo webmaster, que efetivamente desenvolve as classes, usando C++.

Em muitos sistemas, é necessário ainda uma terceira camada de banco de dados, que é um produto separado. Numa simplificação da abordagem, considera-se apenas os bancos de dados relacionais. Alguns dos produtos de banco de dados relacionais existentes são: Oracle, MS SQL server, DB2, Informix, MySql, PostgreSQL. Atualmente, uma forma de se desenvolver sistemas na web em C++ é acessar o produto de banco de dados que se quer por uma API específica para o banco de dados escolhido. Quase todos os produtos de banco de dados oferecem adicionalmente uma biblioteca para permitir o desenvolvimento de aplicações externas que acessem-no. Quase sempre uma das possibilidades de linguagem é C++. O problema é que cada produto de banco de dados possui uma API própria, e incompatível com as demais. Portanto, o código que se escreve para associar o seu código com o banco de dados não é portátil no caso de se querer trocar de banco de dados. Em futuro breve, com uma biblioteca de middleware que está em desenvolvimento - VBCSsql - , a camada de regras de negócio se isolará da camada de banco de dados. Todos os bancos de dados relacionais serão vistos pela camada de regras de negócio da mesma forma.

C++ Multiplataforma e Orientação a Objetos

Parte 6: Programação Genérica (template)

15 Programação genérica (template)

A palavra `template` existe desde a versão 3 da linguagem C++ (não confundir com versão de algum compilador). Todos os compiladores modernos dão suporte a `template`, que pode ser traduzido como *fôrma*²⁹, molde, gabarito. O uso da palavra reservada `template` é um recurso muito versátil que permite a criação de funções e classes de tipo genérico, de tal forma que se pode criar vários modelos repetidos quase idênticos.

Com `template`, pode-se desenvolver 2 tipos de entidades para programação genérica:

- Funções genéricas
- Classes genéricas

Uma função ou classe genérica é “pré-compilada” em genérico. Quando se aplica a classe ou função um “sabor”, então essa classe ou função é “pós-compilada com o sabor em questão”. Tanto a pré-compilação quanto a pós-compilação ocorrem em tempo de compilação, e como todo arquivo `*.cpp` deve ser possível a compilação numa única passagem.

15.1 Analogia entre `template` e *fôrma de bolo*

Uma classe ou função genérica (com `template`) são como se fossem “*fôrmas*” de classe ou função. Como se fossem “*fôrmas de bolo*”. Após definida a *fôrma*, é permitido que se use a *fôrma* com diversos “sabores de bolo”. Sendo que um “sabor de bolo” é ele próprio um “bolo”. Em outras palavras, pode-se aplicar um argumentos de tipo para a classe ou função genérica. Após ter-se aplicado um “sabor ao bolo”, as classes ou funções genéricas tornam-se classes ou funções perfeitamente definidas, isto é, classes ou funções “normais”.

²⁹ Na grafia moderna esta palavra perdeu o acento, porém a forma antiga de grafia foi mantida para diferenciar de *fôrma* (também sem acento).

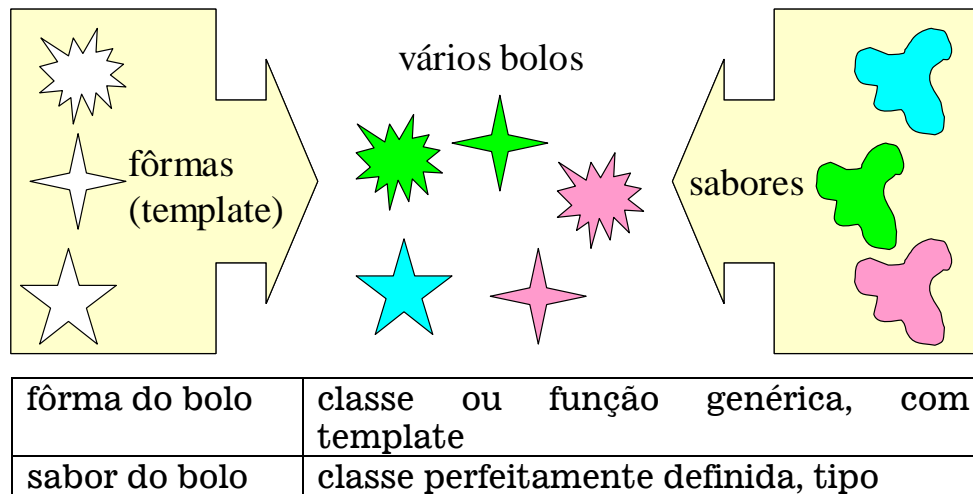


Figura 44: Analogia entre template e fôrma de bolo

15.2 Programação genérica por exemplos

15.2.1 Função genérica

No exemplo abaixo, é feita uma função global genérica swap. Essa função troca os valores de dois argumentos do mesmo tipo, sem saber que tipo de argumento é. Ao compilar a função swap, como trata-se de uma função genérica, o compilador está na realidade pré-compilando a função. Na função main, que utiliza a função genérica swap, apenas por chama-la com argumentos do tipo int, o compilador deduz que trata-se da função com “sabor int”. Com isso, o compilador pós-compila a função swap com int. Caso se deseje evitar que o compilador deduza qual é o “sabor” da função genérica, pode-se explicitamente definir esse argumento: swap<int>. Para ilustrar que a função swap é genérica, na sequência chama-se a função swap com “sabor” float. O sabor de uma classe ou função genérico não necessariamente precisa ser um tipo padrão. Classes ou tipos do usuário podem ser aplicados a classes ou funções genéricas normalmente. O programa abaixo ilustra no final a função swap aplicada a objetos do tipo myStruct (embora não os exiba no console posteriormente).

```
#include <iostream.h>

// generic global function
template <class T>
void swap(T & a, T & b)
{
    T temp = a;
    a = b;
    b = temp;
};

struct myStruct
{
    int m_i;
```

```

double m_d;
myStruct(int i = 0, double d = 0)
{
    m_i = i;
    m_d = d;
}

};

int main()
{
    int x = 3, y = 4;
    swap(x, y); // swap<int>(x,y);
    cout << "x=" << x << " "; y = " << y << endl;
    float f1 = 3.3, f2 = 4.4;
    swap(f1, f2); // swap<float>(f1,f2);
    cout << "f1=" << f1 << " "; f2 = " << f2 << endl;
    myStruct s1 = myStruct(1,2.2), s2 = myStruct(3,4.4);
    swap(s1, s2); // swap<myStruct>(s1,s2);
    return 0;
}

```

A saída do programa é como mostrado abaixo

```

x=4;    y = 3
f1=4.4; f2 = 3.3

```

15.2.2 Classe genérica

Uma classe genérica é uma classe que possui pelo menos um argumento genérico, isto é, há um tipo marcado com template (o sabor do bolo). Enquanto se escreve a classe, não se sabe que tipo será aplicado no lugar do tipo genérico. No exemplo abaixo, a classe genérica myClass possui um genérico que no caso é chamado de T. Trata-se de uma classe simples, que tem um atributo do tipo genérico, e dois métodos: um para definir o valor do atributo e outro para retornar esse valor. O programa principal instancia a classe myClass com o “sabor” int (um tipo padrão) e com myStruct (um tipo do usuário).

```

template <class T>
class myClass
{
    T m_data;
public:
    void set_data(const T & data)
    {
        m_data = data;
    }
    T get_data() { return m_data; }
}; // end of generic class

struct myStruct
{
    int m_i;
    double m_d;
    myStruct(int i = 0, double d = 0)
    {
        m_i = i;
        m_d = d;
    }
};

```

```

int main()
{
    myClass<int> a;
    a.set_data(4);
    int i = a.get_data();
    myClass<myStruct> b;
    b.set_data(myStruct(1, 2.2));
    myStruct x = b.get_data();
    return 0;
}

```

Esse programa não gera saída no console.

No exemplo abaixo, está definida uma classe genérica myMatrix33, que é uma matriz de dimensão fixa 3×3. O tipo de dados que a matriz armazena é genérico. A classe myMatrix33 é a “fôrma do bolo”. A partir de sua definição pode-se criar vários “bolos com fôrmas iguais e sabores diferentes”. No caso os “sabores” são os tipos int, float, double, long double e d_complex. Na classe myMatrix33, o único método definido é operator(), que retorna um lvalue do tipo genérico de dado da matriz. Assim, os objetos das classes derivadas de myMatrix33 poderão usar o operator() de ambos os lados do sinal de =.

Existe mais de uma implementação de suporte a números complexos na biblioteca padrão de C++. A implementação mais sofisticada é a que está sendo usada no exemplo abaixo, que requer o uso de namespace std. Nessa implementação, há uma classe genérica complex que deve ser instanciada com o tipo que o usuário desejar (float, double, long double, por exemplo). O tipo complexo é definido usando o identificador d_complex, que é a instancição da classe genérica complex (definido em #include<complex>), instanciado com o “sabor” double. Uma vez definido o tipo d_complex (a partir da classe genérica complex), esse tipo pode ser usado ele próprio como “sabor” para a classe myMatrix33, criando um novo tipo c_matrix.

```

#include <complex>    // complex
#include <iostream>
using namespace std;

template <class T>
class myMatrix33
{
    T v[3][3];
public:
    T & operator()(int i, int j)
    {
        return v[i][j];
    };
}; // end of generic class myMatrix33

// complex type is defined in #include<complex> in generic
// an usable complex type needs instantiation
typedef complex<double> d_complex;

// defining
typedef myMatrix33 <int>          i_matrix;
typedef myMatrix33 <float>       f_matrix;
typedef myMatrix33 <double>      d_matrix;

```

```

typedef myMatrix33 <long double> ld_matrix;
typedef myMatrix33 <d_complex>    c_matrix;

int main()
{
    // one object of each type
    i_matrix  i_matObj;
    f_matrix  f_matObj;
    d_matrix  d_matObj;
    ld_matrix ld_matObj;
    c_matrix  c_matObj;

    // test usage of objects, by placing some data in any matrix position
    i_matObj(0, 0) = 3;
    cout << i_matObj(0, 0) << endl;
    f_matObj(1, 1) = 4.44444;
    cout << f_matObj(1, 1) << endl;
    d_matObj(1, 1) = 1.11111;
    cout << d_matObj(1, 1) << endl;
    ld_matObj(2, 2) = 2.22222;
    cout << ld_matObj(2, 2) << endl;
    c_matObj(1, 1) = d_complex(1.1, 6.6);
    cout << c_matObj(1, 1) << endl;
    return 0;
}

```

A saída do programa é como mostrado abaixo

```

3
4.44444
1.11111
2.22222
(1.1,6.6)

```

15.2.3 Algoritmos genéricos

Uma grande aplicação de programação genérica é a capacidade de se escrever algoritmos em genérico. Por exemplo: seja o famoso algoritmo de ordenação conhecido como “bolha”, ou *bubble sort*. No exemplo abaixo, a função global genérica `bubble_sort` ordena um array de um tipo genérico com esse algoritmo. Repare que a função global que é o algoritmo de bolha chama o `operator>` da classe genérica. Como não se sabe no momento de escrever o algoritmo qual seria a classe genérica, caso se aplique um “sabor” ao algoritmo de bolha que não tenha o `operator>` definido, então haverá um erro de compilação (erro de pós-compilação).

```

#include <iostream.h>
#include <math.h>

template <class T>
void bubble_sort(T* array, int n)
{
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (array[i] > array[j]) // array[i].operator>(array[j])
            { // swap (array[i], array[j])
                T temp = array[i];
                array[i] = array[j];
                array[j] = temp;
            }
}

```

```
};

int main()
{
    const int N = 10;
    float array[N];
    int i;
    // load random data and show
    cout << "=== Before sort" << endl;
    for (i = 0; i < N; i++)
    {
        array[i] = 10*sin(40*i);
        cout << array[i] << endl;
    }

    // sort with bubble sort
    bubble_sort(array, N);

    cout << "=== After sort" << endl;
    for (i = 0; i < N; i++)
    {
        cout << array[i] << endl;
    }
    return 0;
}
```

A saída do programa é como mostrado abaixo

```
=== Before sort
0
7.45113
-9.93889
5.80611
2.19425
-8.73297
9.45445
-3.87809
-4.28155
9.58916
=== After sort
-9.93889
-8.73297
-4.28155
-3.87809
0
2.19425
5.80611
7.45113
9.45445
9.58916
```

Pode-se escrever uma nova versão do algoritmo de ordenação de bolha, colocando-se mais um argumento, que é um ponteiro para função de comparação. Para usar essa nova versão é preciso que se escreva uma função de comparação, no caso uma função global chamada de myStructCompare.

```
#include <iostream.h>
#include <math.h>

template <class T>
void bubble_sort(T* array, int n, bool(compare))(const T &, const T &))
{
```

```

        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++)
                if (compare(array[i], array[j]))
                { // swap (array[i], array[j])
                    T temp = array[i];
                    array[i] = array[j];
                    array[j] = temp;
                }
    };

struct myStruct
{
    int m_i;
    double m_d;
    myStruct(int i = 0, double d = 0)
    {
        m_i = i;
        m_d = d;
    }
    friend ostream & operator <<(ostream & s, const myStruct & obj)
    {
        s << obj.m_i << " "; " << obj.m_d;
        return s;
    }
};

bool myStructCompare(const myStruct & x, const myStruct & y)
{
    return x.m_d < y.m_d;
}

int main()
{
    const int N = 10;
    myStruct array[N];
    int i;

    // load random data and show
    cout << "=== Before sort" << endl;
    for (i = 0; i < N; i++)
    {
        array[i].m_i = i;
        array[i].m_d = 10*sin(40*i);
        cout << array[i] << endl;
    }

    // sort with bubble sort
    bubble_sort<myStruct>(array, N, myStructCompare);

    cout << "=== After sort" << endl;
    for (i = 0; i < N; i++)
    {
        cout << array[i] << endl;
    }
    return 0;
}

```

A saída do programa é como mostrado abaixo

```

=== Before sort
0; 0
1; 7.45113
2; -9.93889

```

```

3; 5.80611
4; 2.19425
5; -8.73297
6; 9.45445
7; -3.87809
8; -4.28155
9; 9.58916
=== After sort
9; 9.58916
6; 9.45445
1; 7.45113
3; 5.80611
4; 2.19425
0; 0
7; -3.87809
8; -4.28155
5; -8.73297
2; -9.93889

```

Uma outra variação interessante do algoritmo de bolha é deixar efetuar a comparação do algoritmo a partir de uma função global, e passar para um método (membro da classe que está sendo ordenada). No exemplo abaixo, o algoritmo de bolha é feito a partir de um ponteiro para método da classe genérica. Note que quando se passa o parâmetro que é o ponteiro para o método, é preciso que se escolha de qual dos elementos do array de myStruct se vai passar o ponteiro para método. Qualquer um que se escolha dá resultado idêntico, pois o método em si é o mesmo. No exemplo abaixo, escolheu-se o primeiro elemento do array (elemento zero).

```

#include <iostream.h>
#include <math.h>

template <class T>
void bubble_sort(T* array, int n, bool(T::*compare)(const T &))
{
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (array[i].compare(array[j]))
                { // swap (array[i], array[j])
                    T temp = array[i];
                    array[i] = array[j];
                    array[j] = temp;
                }
};

struct myStruct
{
    int m_i;
    double m_d;
    myStruct(int i = 0, double d = 0)
    {
        m_i = i;
        m_d = d;
    }
    friend ostream & operator <<(ostream & s, const myStruct & obj)
    {
        s << obj.m_i << " "; " << obj.m_d;
        return s;
    }
}

```

```

    bool compare(const myStruct & x)
    {
        return m_d < x.m_d;
    }
};

int main()
{
    const int N = 10;
    myStruct array[N];
    int i;

    // load random data and show
    cout << "=== Before sort" << endl;
    for (i = 0; i < N; i++)
    {
        array[i].m_i = i;
        array[i].m_d = 10*sin(40*i);
        cout << array[i] << endl;
    }

    // sort with bubble sort
    bubble_sort<myStruct>(array, N, array[0].compare);

    cout << "=== After sort" << endl;
    for (i = 0; i < N; i++)
    {
        cout << array[i] << endl;
    }
    return 0;
}

```

A saída do programa é o mesmo do exemplo anterior.

15.3 Classes genéricas contenedoras

Uma aplicação muito útil de classe genérica são as chamadas classes do tipo “contenedora”, isto é, uma classe feita para conter dados. Há várias estruturas adequadas para se conter dados. Uma delas é a lista encadeada. A lista encadeada genérica é desenvolvida sem que se saiba o tipo do dado do qual se está armazenando. Outros tipos de classe contenedora genérica são vetores (vector), fila (queue), fila com dupla terminação (*double ended queue*, ou deque), entre outras.

No módulo 6, discute-se STL – Standard Template Library – uma extensão da biblioteca padrão de C++, toda escrita em genérico.

15.3.1 VList: uma lista encadeada genérica

Nessa sub-seção, descreve-se a classe VList (encontrada em VLib: www.vbmegi.org/vbllib), que é uma lista encadeada genérica simplesmente encadeada, com alguns recursos. Dentro da classe VList há uma estrutura chamada node (nó), que é a estrutura que é replicada na lista. Uma lista encadeada é um conjunto de “nodes” um apontando para o outro. Há também

alguns atributos do objeto lista, que armazenam informações importantes, tal como ponteiro para o primeiro e para o último node da lista.

A interface da VBList contém os seguintes métodos:

- construtor implícito (*default constructor*), e destrutor.
- construtor de cópia e operator= (para garantir que um objeto do tipo VBList possa ser passado por valor como argumento para uma função, se for necessário).
- GetFirst e GetNext, usados para varrer a lista.
- add, para acrescentar um elemento na lista, usando operator=.
- remove, para remover todos os elementos da lista que retornam verdadeiro com o operator==.
- deleteAll, para remover toda a lista.

O código fonte da VBList está mostrado abaixo.

```
// linked list template class
template <class dataType>
class VBList
{
    class node
    {
    public:
        dataType data;
        node *p_next;
    }; // end of class node

    node *p_first, *p_last, *p_nextSave;
    // void (*error_function)(); // pointer to function

public:
    // default constructor
    VBList()
    {
        p_first = p_last = p_nextSave = NULL;
    }; // end of VBList();

    dataType *GetFirst()
    {
        if (p_first)
        {
            p_nextSave = p_first->p_next;
            return &p_first->data;
        }
        else
            return NULL;
    };

    dataType *GetNext()
    {
        node *ret = p_nextSave;
        if (p_nextSave)
        {
            p_nextSave = p_nextSave->p_next;
        }
    }
};
```

```

        return &ret->data;
    }
    else
        return NULL;
};

// add data to list (addTail)
bool add(const dataType & newData)
{
    node *p_new;
    // ASSERT(p_new=new node);
    if ((p_new = new node) == NULL)
        return (1);

    p_new->data = newData; // copies data to list
    // if new element is the first element of list
    if (!p_first)
    {
        p_first = p_last = p_new;
        p_new->p_next = NULL;
    }
    // if new element is NOT the first element of list
    else
    {
        p_last->p_next = p_new; // previous p_last points to new element
        p_new->p_next = NULL; // finish list
        p_last = p_new; // now p_last points to new element
    }
    return (0);
}; // end of void add(dataType & newData)

// delete the whole list
void deleteAll()
{
    node *p_t1, *p_t2;
    if (p_first) // if list exists
    {
        p_t1 = p_first;
        p_t2 = p_first->p_next;
        do
        { // since list exists, at least 1 delete
            delete p_t1;
            p_t1 = p_t2;
            if (p_t2)
                p_t2 = p_t2->p_next;
        } while (p_t1 != NULL);
        p_last = p_first = NULL;
    }
}; // end void deleteAll()

// delete one data from list
void remove(const dataType & dataToDelete) {
    node *p_t1, *p_t2, *p_t3;
    if (p_first) // if list exists
    {
        p_t1 = p_first;
        p_t2 = p_first->p_next;

        // if data to delete is the first one
        // dataType must have operator== defined
        if (p_first->data == dataToDelete) {

```

```

        // for debug
        // cout << "DEBUG Deleted:"<< p_first->data << endl;
        delete p_first;
        p_first = p_t2;
        remove(dataToDelete); // recursively calls remove, to
        // be sure
    }
    else { // the data to delete is not the first one

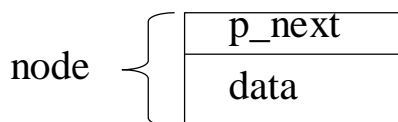
        // this loop will seek for data to delete and delete it
        while (p_t1->p_next != NULL)
        { // since list exists, at least 1 delete

            if (p_t2->data == dataToDelete) {
                // for debug
                // cout << "DEBUG Deleted:"<< p_t2->data << endl;
                p_t3 = p_t2->p_next;
                delete p_t2;
                p_t1->p_next = p_t3; // re-links the list (bypass p_t2)
                p_t2 = p_t3; // to keep going
            }
            else {
                // move pointers one step into the list
                // only if no remove was made.
                p_t1 = p_t2;
                p_t2 = p_t2->p_next;
            }
        } // while
    } // else
} // if p_first
} // remove

// default destructor
~VBList()
{
    deleteAll();
}; // end of ~VBList();
}; // end of template <class dataType>

```

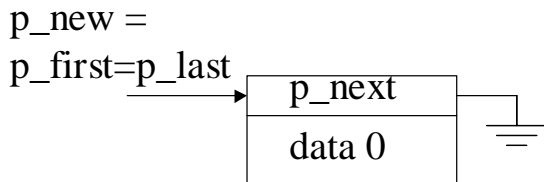
A lista é um encadeamento de nodes, sendo que um node é uma classe que contém 2 campos. Um é um ponteiro para node, que é carregado com o endereço do próximo node da lista (se for o último da lista usa-se o valor NULL, isto é, zero). A visualização gráfica de um node está mostrado na figura abaixo.



Após o construtor, `p_first` e `p_last` estão inicializados com NULL (zero). O método `add` faz a lista alocar espaço para mais um dado e encadea-lo aos demais. Para fazer isso o método `add` aloca um node e faz uma variável local tipo (ponteiro para node) chamada `p_new` apontar para o novo node.

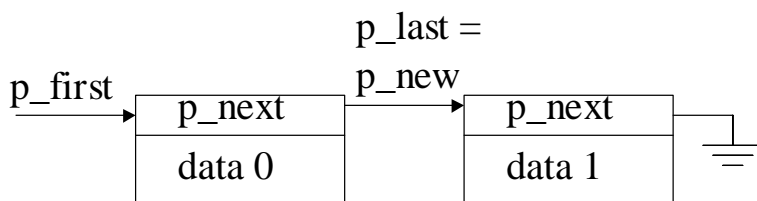
Se o node alocado for o primeiro de todos (data 0), os dados membro da classe `VBList` chamados `p_first` e `p_last` serão inicializados apontando para esse único

node. O campo `p_next` desse node será inicializado com NULL e o dado será copiado para o campo de dados do node.

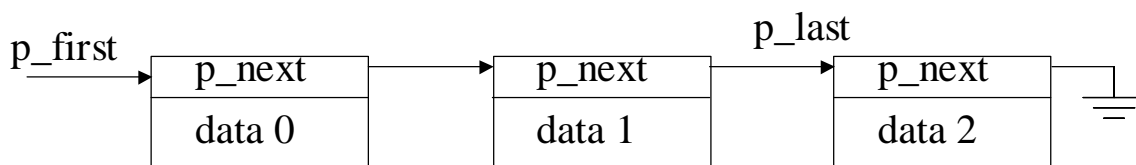


Com a instrução `p_new->data = newData;` copia-se os dados para o campo `data`. Lembre-se que os dados são copiados de forma genérica, isto é, com `template`. Quando se escreve a classe de lista, que nesse caso chama-se `VList`, não se sabe a priori qual é o método que copia os dados, justamente porque a classe que chama esse método é genérica (`template`). Apenas sabe-se que para que uma classe de entrada que seja usada nessa lista, é preciso que o método `operator=` esteja definido. Ao efetuar a compilação, mesmo que não haja erro de sintaxe na classe `VList` (a forma do bolo), pode surgir um erro de compilação quando se aplicar nessa lista uma classe de entrada (sabor do bolo) que não tenha o método `operator=` definido, ou que tenha bug nesse método.

Com a entrada do próximo dado na lista (data 1), o método `add` aloca novo node, faz o ponteiro que era o último da lista (nesse caso o ponteiro do node 0), apontar para o novo node, faz o próximo do node novo apontar para zero e atualiza a variável membro da classe `VList` chamada `p_last` para que aponte para o novo último node da lista.



Chamando mais uma vez o método `add` (data 2), a lista ficará com a representação gráfica como mostrado na figura abaixo. Nessa representação, não é mostrado a variável temporária `p_new`.



Por várias razões pode-se querer varrer a lista desde o primeiro elemento até o último. Isso pode ser feito com o laço de programação mostrado abaixo, que usa um ponteiro para `dataType` e os métodos `GetFirst` e `GetNext`. No exemplo abaixo, o laço varre a lista e coloca cada elemento no console, usando o operador `insersor (<<)`.

```

void g() {
    typedef int TYPE;
    VBList<TYPE> listObj;
    // add data to list
    TYPE *p; // a pointer to dataType
    for (p = listObj.GetFirst() ; p ; p = listObj.GetNext() ) {
        cout << *p << endl;
    }
}

```

Suponha que se deseje procurar um elemento da lista com uma característica específica, digamos cujo conteúdo do campo “telephone” seja igual a “222-2222”. Essa busca pode ser feita fazendo a comparação em questão dentro do loop que varre a lista.

No exemplo abaixo, definiu-se uma classe chamada `data_tel_name`, que contém 2 campos de dados do tipo `VBString`, contendo nome e telefone. Para essa classe, definiu-se o `operator=` e `operator==`, além de alguns outros métodos. Para simplificar, definiu-se um construtor com 2 parâmetros, que permite fazer a entrada de dados diretamente (sem a necessidade de um objeto separado).

```

#include "vblib.h" // VBString, VBList

// the data class must have operator= and operator== defined
class data_tel_name {
    VBString m_telephone, m_name;
public:
    // default constructor
    data_tel_name() {
        m_name = "no name";
        m_telephone = "no telephone";
    };

    // constructor with parameters
    data_tel_name(VBString name, VBString tel) {
        m_name = name;
        m_telephone = tel;
    };

    VBString getName () { return m_name; }
    VBString getPhone () { return m_telephone; }

    void operator=(const data_tel_name & a) {
        m_telephone = a.m_telephone;
        m_name = a.m_name;
    };

    // return true if a==inner. return false if a!=inner
    // compares only telephone, ignores other fields
    bool operator==(const data_tel_name & a) {
        return m_telephone == a.m_telephone;
    };
}; // end of data_tel_name

int main () {
    VBList<data_tel_name> listObj; // listObj is instance of myTelNameList

    // add some data
    listObj.add(data_tel_name("Sergio","222-2222"));
    listObj.add(data_tel_name("Marcia","333-3333"));
}

```

```

listObj.add(data_tel_name("Camila","444-4444"));
listObj.add(data_tel_name("Mateus","555-5555"));

VBString searchPhone;
cout << "Enter telephone to search: ";
cin >> searchPhone;

data_tel_name *p; // a pointer to the data type
bool found;
// this loop scans the entire list
for (p = listObj.GetFirst() ; p ; p = listObj.GetNext() ) {
    data_tel_name a = data_tel_name("",searchPhone);
    found = (a == *p);
    if (found) break;
}

if (found)
    cout << "The name of person with telephone \"" << p->getPhone() <<
        "\" is \"" << p->getName() << "\"<< endl;
else
    cout << "No person was found with telephone \"" <<
        searchPhone << "\"<< endl;
return 0;
}

```

Nesse programa, um objeto do tipo lista é criado, e alguns dados são acrescentados à lista. Em seguida, pergunta-se por um telefone. O laço de varredura da lista é usado para procurar elemento a elemento por um retorno válido do operador== (que por sua vez, compara apenas o campo de telefone). Caso o elemento seja encontrado, termina-se a procura. Caso contrário, faz-se a varredura até o fim.

Após a varredura, mostra-se o que foi encontrado ou informa-se que o elemento não foi encontrado. Um exemplo de saída poderia ser como abaixo.

```

Enter telephone to search: 222-2222
The name of person with telephone "222-2222" is "Sergio"

```

Outro exemplo poderia ser como abaixo.

```

Enter telephone to search: 222-3333
No person was found with telephone "222-3333"

```

15.4 VBMath - uma biblioteca de matemática matricial em genérico

Um exemplo interessante de uso de classe genérica é a elaboração de classes para manipulação de matrizes (ou vetores). Nessa seção, será mostrado o funcionamento de uma biblioteca de manipulação matemática matricial em programação genérica (usando template). Essa biblioteca chama-se VBMath (Villas-Boas math), e pode ser encontrada na Internet no endereço abaixo.

<http://www.vbmcgi.org/vbmath/>

Na listagem abaixo, está mostrado um esqueleto de uma classe de matriz usando um parâmetro como classe genérica.

```

template <class REAL>

```

```

class vb_matrix {
    // member data
    REAL *data;
    unsigned rows;    // number of rows.
    unsigned cols;    // number of columns.
public:
    // methods ...
};

```

Uma classe de matriz definida assim é uma “forma de bolo”, cujos “sabores” poderão ser por exemplo: float, double, long double, etc. Aplicando-se os sabores a forma do bolo, cria-se um tipo para cada sabor. Isso pode ser feito com o código abaixo.

```

typedef vb_matrix<long double> vb_longDoubleMatrix;
typedef vb_matrix<double> vb_doubleMatrix;
typedef vb_matrix<float> vb_floatMatrix;

```

Tendo definido os tipos, pode-se instanciar os objetos e usa-los, com suas funções membro. Abaixo estão listados alguns exemplos didáticos de uso da VBmath. O primeiro exemplo é a simples alocação criação de uma matriz de double, com dimensão 2×2. Essa matriz é carregada com constantes e em seguida é exteriorizada pela stream diretamente em cout. A dimensão da matriz é passado para o objeto como parâmetro. O objeto `M` do tipo `vb_doubleMatrix` aloca a memória necessária para os dados no construtor. O objeto `M` pode usar normalmente os dados até que seja terminado o seu escopo, quando será chamado o destrutor, que libera a memória alocada. Entre outros, está definido o método `operator()`, que retorna um lvalue do tipo `REAL &`, sendo que `REAL` é o tipo genérico (template) de elemento da matriz. Por isso, pode-se acessar os elementos da matriz de ambos os lados de um operador de designação (*assignment operator*). O método `operator<<` contém informações que permitem um objeto da classe ser colocado numa stream.

```

// vb_matrix (unsigned rows, unsigned cols=1);
// friend ostream & operator<< (ostream & stream, vb_matrix<REAL> & obj);
#include "vbmath.h" // main include file

int main() {
    int s=2;
    vb_doubleMatrix M(s,s);
    M(0,0) = 1.1;
    M(0,1) = 2.1;
    M(1,0) = 3.1;
    M(1,1) = 4.1;
    cout << M;
    return 0;
}

```

Resultado

```

(2,2)
+1.1000 +2.1000
+3.1000 +4.1000

```

No exemplo abaixo, demonstra-se o método `operator=`, que faz a cópia elemento a elemento do objeto `M` para o objeto `N`. O resultado é o mesmo anterior. No caso da VBmath, o método `operator=` retorna void (isto é, não retorna).

```
// void operator=(vb_matrix<REAL> & x);
#include "vbmath.h" // main include file
```

```
int main() {
    vb_doubleMatrix M(2,2);
    M(0,0) = 1.1;
    M(0,1) = 2.1;
    M(1,0) = 3.1;
    M(1,1) = 4.1;
    vb_doubleMatrix N=M;
    cout << N;
    return 0;
}
```

Em C/C++ o método `operator=` padrão retorna o mesmo tipo da classe. Com isso, é possível escrever `a=b=c=d;` por exemplo. Por motivo de desempenho, não é recomendável que o método `operator=` retorne dessa forma no caso de uma biblioteca matricial.

No exemplo abaixo, demonstra-se o método `operator+()`, nesse caso somando a matriz com um `REAL` (o mesmo tipo do elemento da matriz). Nesse caso, a soma é feita elemento a elemento.

```
// friend vb_matrix<REAL> operator+(vb_matrix<REAL> & mat, REAL re);
#include "vbmath.h" // main include file
```

```
int main() {
    vb_doubleMatrix M(2,2);
    M(0,0) = 1;
    M(0,1) = 2;
    M(1,0) = 3;
    M(1,1) = 4;
    vb_doubleMatrix N=M+1.1;
    cout << N;
    return 0;
}
```

Resultado

```
(2,2)
+2.1000 +3.1000
+4.1000 +5.1000
```

No exemplo abaixo, demonstra-se o método `operator*`, nesse caso multiplicando a matriz com um `REAL` (o mesmo tipo do elemento da matriz). Nesse caso, a multiplicação é feita elemento a elemento.

```
// friend vb_matrix<REAL> operator*(vb_matrix<REAL> & mat, REAL re);
#include "vbmath.h" // main include file
```

```
int main() {
    vb_doubleMatrix M(2,2);
    M(0,0) = 1.1;
    M(0,1) = 2.1;
    M(1,0) = 3.1;
    M(1,1) = 4.1;
    vb_doubleMatrix N=M*2.5;
    cout << N;
    return 0;
}
```

Resultado


```
(2,2)
+2.7500 +5.2500
+7.7500 +10.2500
```

No exemplo abaixo, demonstra-se o método `operator*`, nesse caso multiplicando a matriz com outra matriz. Nesse caso, a operação é a multiplicação matricial seguindo as normas matemáticas. O retorno da função é um objeto tipo matriz com as dimensões adequadas. Caso a operação de multiplicação matricial seja impossível por problema de dimensão dos argumentos, um erro de execução é gerado.

A resposta da multiplicação é armazenado no objeto `K1`, criado diretamente, ou no objeto `K2` criado inicialmente e posteriormente recebendo o retorno da função. A tentativa de multiplicar as matrizes de forma incompatível com suas dimensões resulta em erro, tratado pela própria biblioteca.

```
// vb_matrix<REAL> operator* (vb_matrix<REAL> & arg);
#include "vbmh.h" // main include file

int main() {
    vb_doubleMatrix M(2,2);
    M(0,0) = 1.1;
    M(0,1) = 2.1;
    M(1,0) = 3.1;
    M(1,1) = 4.1;
    vb_doubleMatrix N(2,1);
    N(0,0) = 3.3;
    N(1,0) = 7.3;
    vb_doubleMatrix K1=M*N;
    vb_doubleMatrix K2; K2=M*N;
    cout << M << N << K1 << K2;
    vb_doubleMatrix J=N*M; // execution error, incompatible simensions
    return 0;
}
```

Resultado

```
(2,2)
+1.1000 +2.1000
+3.1000 +4.1000
```

```
(2,1)
+3.3000
+7.3000
```

```
(2,1)
+18.9600
+40.1600
```

```
(2,1)
+18.9600
+40.1600
```

```
=====
VBmath - The Villas-Boas library for mathematics in C++ multiplatform
Version 1.0 of October 27, 1999
http://www.del.ufrj.br/~villas/cpplib/vbmath/
Error:arg1.cols must be equal to arg2.rows to perform matrix multiplication
=====
```

No exemplo abaixo, demonstra-se o método `operator+`, nesse caso somando a matriz com outra matriz. Também mostra-se que um objeto matriz pode ser atribuído seguidamente sem problemas. Na primeira vez que o objeto recebe uma atribuição, aloca-se memória para os dados. Na vez seguinte que o objeto recebe atribuição, a memória é liberada e nova memória é alocada, tudo automaticamente.

```
// vb_matrix<REAL> operator+ (vb_matrix<REAL> & arg);
#include "vbmath.h" // main include file

int main() {
    vb_doubleMatrix M(2,2);
    M(0,0) = 1.5;
    M(0,1) = 2.5;
    M(1,0) = 3.5;
    M(1,1) = 4.5;
    vb_doubleMatrix N(2,2);
    N(0,0) = 1.1;
    N(0,1) = 2.2;
    N(1,0) = 3.3;
    N(1,1) = 4.4;
    vb_doubleMatrix K=M-N;
    K=M+N;
    cout << M << N << K;
    return 0;
}
```

Resultado

```
(2,2)
+1.5000 +2.5000
+3.5000 +4.5000
```

```
(2,2)
+1.1000 +2.2000
+3.3000 +4.4000
```

```
(2,2)
+2.6000 +4.7000
+6.8000 +8.9000
```

O exemplo abaixo demonstra o método `inv`, que calcula a inversa da matriz. A matriz M multiplicada pela sua inversa K é a matriz identidade J .

```
#include "vbmath.h" // main include file

int main() {
    vb_doubleMatrix M(2,2);
    M(0,0) = 1.1;
    M(0,1) = 2.1;
    M(1,0) = 3.1;
    M(1,1) = 4.1;
    vb_doubleMatrix K;
    K=M.inv();
    vb_doubleMatrix J=K*M;
    cout << M << K << J;
    return 0;
}
```

Resultado

```
(2,2)
```

```
+1.1000 +2.1000
+3.1000 +4.1000
```

```
(2,2)
-2.0500 +1.0500
+1.5500 -0.5500
```

```
(2,2)
+1.0000 +0.0000
+0.0000 +1.0000
```

O exemplo abaixo tem uma função `t`, cujo parâmetro é a dimensão de uma matriz quadrada. A função preenche a matriz com dados aleatórios e calcular a inversa dessa matriz. Em seguida, multiplica a matriz original pela sua inversa (o que gera a matriz identidade com dimensão igual ao parâmetro de entrada). Em seguida, mostra as 3 matrizes.

```
#include "vbmath.h" // main include file

void t(unsigned size=3) {
    unsigned i,j;
    vb_longDoubleMatrix M(size,size);
    for ( i = 0; i < size; ++i )
        for ( j = 0; j < size; ++j )
            M(i,j) = rand()/10000.0;
    vb_longDoubleMatrix K;
    K=M.inv();
    vb_longDoubleMatrix J=K*M;
    cout << M << K << J;
}

int main() {
    t(5);
}
```

Resultado

```
(5,5)
+0.0041 +1.8467 +0.6334 +2.6500 +1.9169
+1.5724 +1.1478 +2.9358 +2.6962 +2.4464
+0.5705 +2.8145 +2.3281 +1.6827 +0.9961
+0.0491 +0.2995 +1.1942 +0.4827 +0.5436
+3.2391 +1.4604 +0.3902 +0.0153 +0.0292
```

```
(5,5)
-0.2580 +0.3976 +0.0471 -0.9724 +0.1225
+0.6938 -0.9660 -0.1837 +2.2124 +0.4669
-0.5682 +0.4117 +0.4011 -0.5699 -0.2611
-1.7145 +2.4306 +1.6755 -7.8903 -1.3533
+2.4118 -2.5664 -2.2720 +8.9669 +1.5070
```

```
(5,5)
+1.0000 +0.0000 +0.0000 -0.0000 -0.0000
-0.0000 +1.0000 -0.0000 +0.0000 -0.0000
+0.0000 -0.0000 +1.0000 -0.0000 -0.0000
+0.0000 +0.0000 +0.0000 +1.0000 +0.0000
+0.0000 -0.0000 -0.0000 +0.0000 +1.0000
```

16 STL - Standard Template Library

16.1 Introdução

Esse capítulo fala de forma resumida sobre STL, ou Standard Template Library - que em português significa algo como “biblioteca padrão genérica”. O termo “genérica” nesse caso significa “programação genérica”, como explicado no capítulo de “//”, na página //. O leitor interessado em STL deve considerar o exposto nesse capítulo apenas como uma introdução, e procurar referências mais completas para uma leitura mais aprofundada.

Para quem usa Visual C, procure por exemplos de STL procurando no help por “STL sample programs”.

STL é atualmente considerada como parte da biblioteca padrão de C++. A idéia básica do STL é implementar um conjunto de componentes típicos com template, de forma a que esses componentes possam ser usados de forma genérica. STL dá ênfase no uso de classes contenedoras (*container*) e iteradoras (*iterator*)³⁰, e também na implementação de algoritmos genéricos. As classes contenedoras são listas encadeadas, vetores, mapas, etc. As classes iteradoras são aquelas que permitem apontar para os elementos das classes contenedoras, e varrer esses elementos. Os algoritmos são find, sort, binary_search, etc.

A biblioteca STL é parte da biblioteca padrão de C++, mas merece atenção separada, por sua estrutura, sua abrangência e sua generalidade. O conceito de programação genérica (com template) é independente do conceito de namespace. A biblioteca STL pode ser usada com ou sem namespace std. Algumas das funcionalidades mais novas e sofisticadas da biblioteca padrão de C++ somente estão implementadas com namespace std (e.g classe de string, novidades na classe complex, etc.). Desde que o uso de STL também é de certa forma um assunto sofisticado, estou assumindo que quem está interessado em STL tem acesso a um compilador C++ moderno que tenha suporte pleno a namespace. Todos os exemplos desse capítulo usam namespace std. Caso o leitor procure referências adicionais sobre STL, com certeza irá encontrar exemplos em que STL é usado sem namespace. Como já foi dito, STL é baseado no conceito de programação genérica (com template), que não tem relação com o conceito de namespace.

³⁰ Em latim, *iterare* significa repetir. Não confundir com “interar” (verbo que não existe, mas que lembra “interação”, ou “interativo”).

16.1.1 Classes contenedoras

STL provê diferentes tipos de classes contenedoras, sempre formuladas com template para que possam ser usadas de forma genérica. As classes contenedoras - `vector`, `list`, `map` - tem um método chamado `size()`, por exemplo, que retorna o número de elementos que estão contidos na classe contenedora. Há também outros métodos, tais como `begin()` e `end()`, que são usados para localizar o primeiro elemento e a primeira posição *após* o último elemento. Uma classe contenedora vazia possui valores iguais para o retorno dos métodos `begin()` e `end()`.

16.1.2 Classes iteradoras

Os iteradores são como ponteiros. São usados para acessar os elementos de uma classe contenedora. Os iteradores podem mover-se de um elemento para o outro, sendo que a implementação do código dessa movimentação é implícita para quem usa os iteradores. Por exemplo, usando um objeto da classe contenedora `vector`, o operador `++` sobre um objeto iterador significa passar para a posição de memória onde encontra-se o próximo elemento armazenado.

16.1.3 Algoritmos

Os algoritmos implementados funcionam com iteradores acessando objetos de classes contenedoras.

Os objetos de classes contenedoras podem ser acessados por iteradores, e os algoritmos usam os iteradores. Dessa forma, os algoritmos podem ser usados de forma excepcionalmente clara.

contenedores \Leftrightarrow iteradores \Leftrightarrow algoritmos

16.2 Preâmbulo

Para prepararmos-nos para a programação no estilo STL, vamos considerar o exemplo abaixo. No exemplo, cria-se uma classe contenedora com dados do tipo 'int'. Cria-se um iterador para esse tipo. Escreve-se a função `my_find` para encontrar um elemento na classe contenedora. No programa principal enche-se a classe contenedora com dados (números pares). A função `my_find` encontra a posição relativa a ocorrência dos valores entrados pelo usuário.

A elegância desse código está no fato de que o algoritmo de encontrar um valor foi escrito sem que se considere o tipo a ser procurado (no caso 'int'), ou que se considere como o iterador vai para a próxima posição (no caso 'begin++').

```
#include <iostream>
using namespace std;

// creating the new type for Iterator, a pointer to 'int'
typedef int* Iterator;

// implementing a find algorithm. my_find is a global function
```

```

Iterator my_find(Iterator begin, Iterator end, const int & Value) {
    while (begin != end && // pointer comparison
           *begin != Value) // object comparison
        begin++; // next position
    return begin;
}

int main () {
    const int max = 100;
    int aContainer[max]; // the container object.
                        // it contains max elements of type 'int'
    Iterator begin = aContainer; // point to the begin
    Iterator end = aContainer + max; // point to the position after the last element

    // fill the container with some data
    for (int i=0; i < max ; i ++ )
        aContainer[i] = 2*i;

    int Number=0;
    while (1) { // for ever
        cout << "Enter required number (-1 end):";
        cin >> Number;
        if (Number == -1) break;
        Iterator position = my_find(begin,end,Number);
        if (position != end)
            cout << "Found at position " << (position - begin) << endl;
        else
            cout << Number << " not found" << endl;
    }
    return 0;
}

```

Um exemplo do uso do programa é mostrado abaixo.

```

Enter required number (-1 end):12
Found at position 6
Enter required number (-1 end):44
Found at position 22
Enter required number (-1 end):33
33 not found
Enter required number (-1 end):-1

```

Uma variação interessante desse programa seria re-escrever a função `my_find` usando template. Essa variação pode ser feita sem que o resto do programa precise ser alterado.

```

// implementing a find algorithm. my_find is a global function
template <class iteratorType, class valueType>
Iterator my_find(iteratorType begin, iteratorType end, const valueType & Value) {
    while (begin != end && // pointer comparison
           *begin != Value) // object comparison
        begin++; // next position
    return begin;
}

```

No exemplo abaixo, estamos de fato usando o STL. O header ‘algorithm’ é necessário para incluir a definição da função global com template ‘find’, e o header ‘vector’ é necessário para a definição da classe genérica ‘vector’. Para isolar o mais possível o problema do tipo que se está usando nos dados, criou-se uma macro `TYPE`, que significa ‘int’. O tipo ‘Iteractor’ poderia continuar sendo

um ‘TYPE*’, mas é mais elegante usar a definição pelo STL, como mostrado no programa.

Outra coisa boa: o objeto contenedor agora é alocado (e liberado) automaticamente. Portanto, o número de elementos ‘max’ não precisa mais ser um const. Pode ser um int normal (isto é, pode ser definido em tempo de execução; antes não podia).

O objeto aContainer agora é do tipo vector, quando a essa classe genérica (forma do bolo) é aplicado o tipo (sabor do bolo) ‘TYPE’, isto é ‘int’. Ou seja, aContainer é um vetor de int. Esse vetor é carregado com dados da mesma forma que antes, e a forma de usar o find é muito parecido com o my_find anterior.

```
#include <iostream>
#include <algorithm> // STL, find
#include <vector> // STL, vector
using namespace std;
#define TYPE int
// creating the new type for Iterator, a pointer to 'int'
// typedef TYPE* Iterator; // directly define the iterator type
typedef vector<TYPE>::iterator Iterator; // define iterator type through STL

int main () {
    int max = 100;
    vector<TYPE> aContainer(max); // the container object.

    // fill the container with some data
    for (int i=0; i < max ; i ++ )
        aContainer[i] = 2*i;

    int Number=0;
    while (1) { // for ever
        cout << "Enter required number (-1 end):";
        cin >> Number;
        if (Number == -1) break;
        Iterator position = find(aContainer.begin(),aContainer.end(),Number);
        if (position != aContainer.end())
            cout << "Found at position " << (position - aContainer.begin())<<endl;
        else
            cout << Number << " not found" << endl;
    }
    return 0;
}
```

16.2.1 Classes e funções auxiliares

Essa seção descreve algumas ferramentas de STL que serão necessárias para compreensão posterior.

16.2.1.1 Par (pair)

Um par no sentido de STL é o encapsulamento de dois objetos que podem ser de tipos diferentes. Os pares são componentes fundamentais, e que serão usados mais tarde nos exemplos de STL. São definidos como uma struct pública no header <utility>:

```
template <class T1, class T2>
struct pair {
    T1 first;
    T2 second;
    pair(){}; // empty default constructor
    pair(const T1 & a, const T2 & b) // call constructor of parent class
        : first(a), second(b) {};
};
```

O segundo construtor padrão faz os elementos a serem inicializados pelos construtores de seu tipo.

Para a classe pair, OS `operator==` e `operator<` são os comparadores globais definidos no STL.

```
template <class T1, class T2>
inline bool operator==(const pair<T1, T2> & x, const pair<T1, T2> & y) {
    return x.first == y.first && x.second == y.second;
}
```

```
template <class T1, class T2>
inline bool operator<(const pair<T1, T2> & x, const pair<T1, T2> & y) {
    return x.first < y.first ||
        (!(x.first < y.first) && x.second < y.second);
}
```

Em `operator<`, quando o primeiro objeto é igual, o retorno é determinado pela comparação do segundo objeto do par. Contudo, de forma a fazer o mínimo de exigências possível para um objeto poder fazer parte de um par, `operator==` não é usado em `operator<`.

Para facilitar a criação de pares há a função global abaixo.

```
template <class T1, class T2>
inline pair<T1, T2> make_pair(const T1 & x, const T2 & y) {
    return pair<T1, T2>(x,y);
}
```

16.2.2 Operadores de comparação

O `operator!=` é definido tendo como base o `operator==`.

```
template <class T>
inline bool operator != (const T & x, const T & y) {
    return !(x==y);
}
```

O `operator>` é definido tendo como base o `operator<`.

```
template <class T>
inline bool operator> (const T & x, const T & y) {
    return y < x;
}
```

O `operator<=` é definido tendo como base o `operator<`.

```
template <class T>
inline bool operator<= (const T & x, const T & y) {
    return !(y < x);
}
```

O `operator>=` é definido tendo como base o `operator<`.

```
template <class T>
inline bool operator>= (const T & x, const T & y) {
    return y <= x;
}
```



```
return !(x < y);
}
```

Em tese o `operator==` poderia ser definido a partir exclusivamente de `operator<`, conforme abaixo. Portanto, o termo “igualdade” não deveria ser usado, e deveria ser substituído pelo termo “equivalência”. Mas há resistências ao uso do `operator==` dessa forma.

```
// not part of STL
template <class T>
inline bool operator== (const T & x, const T & y) {
return !(x < y) && !(y < x);
}
```

16.2.3 Classes de comparação

STL possui várias classes de comparação, que podem ser usadas para várias finalidades. A tabela abaixo mostra as classes de comparação definidas no header `<functional>`.

Definição do Objeto	Chamada	Retorno
<code>equal_to<T> X;</code>	<code>X(x,y)</code>	<code>x == y</code>
<code>not_equal_to<T> X;</code>	<code>X(x,y)</code>	<code>x != y</code>
<code>greater<T> X;</code>	<code>X(x,y)</code>	<code>x > y</code>
<code>less<T> X;</code>	<code>X(x,y)</code>	<code>x < y</code>
<code>greater_equal<T> X;</code>	<code>X(x,y)</code>	<code>x >= y</code>
<code>less_equal<T> X;</code>	<code>X(x,y)</code>	<code>x <= y</code>

Classes de comparação do STL

A idéia é que essas classes forneçam ponteiros para funções de comparação, de forma que seja fácil (entenda-se: com interface uniforme) escrever uma função que implementa um algoritmo qualquer (que usa comparação). Assim, pode-se escrever um algoritmo que recebe como um dos parâmetros, um ponteiro para a função de comparação. Isso significa que pode-se usar o mesmo algoritmo com a função de comparação que se desejar.

Com a finalidade de padronizar os nomes nas em todas as funções de comparação, STL definiu uma função de comparação base, da qual as demais herdam. Essa função é `binary_function`, mostrada abaixo. Para classes unárias, uma classe correspondente chamada `unary_function` também é definida. O leitor interessado poderá vê-la examinando o header `<functional>`.

```
template <class Arg1, class Arg2, class Result>
struct binary_function {
    typedef Arg1 first_argument_type;
    typedef Arg2 second_argument_type;
    typedef Result result_type;
};
```

A implementação da classe de comparação `equal_to` é exemplificada abaixo. A palavra reservada “const” colocada antes de “{” na terceira linha significa que o ponteiro para função que corresponde ao `operator()` somente poderá ser passado como argumento para uma função que tenha “const” nesse argumento.

```
template <class T>
```

```

struct equal_to : binary_function<T, T, bool> {
    bool operator()(const T & x, const T & y) const {
        return x==y;
    }
};

```

No programa abaixo, implementa-se o famoso algoritmo de ordenação “bolha” (bubble_sort) usando template e com baseado numa função de comparação passada ao algoritmo como ponteiro para função.

A forma com que se escreveu o algoritmo bubble_sort é considerado elegante, pois pode-se usar a mesma função para ordenar um array (qualquer) com a função de comparação que se queira. Tanto pode-se usar as funções de comparação derivadas das classes de comparação do STL, quanto pode-se criar funções de comparação do usuário nos mesmos moldes do STL e usar o mesmo algoritmo de ordenação.

```

#include <iostream>
#include <functional> // less<T>
#include <cstdlib> // abs
using namespace std;

template <class T, class CompareType>
void bubble_sort(T* array, int n, const CompareType & compare) {
    for (int i=0 ; i < n ; i++)
        for (int j=i+1 ; j < n ; j++)
            if (compare(array[i],array[j])) {
                // swap (array[i], array[j])
                T temp = array[i];
                array[i] = array[j];
                array[j] = temp;
            }
}

// my comparison function
template <class T>
struct absolute_less : binary_function<T,T,bool> {
    bool operator()(const T & x, const T & y) const {
        return abs(x) < abs(y);
    }
};

// a global function to help display the array
void display(int *array, int n) {
    for (int i=0 ; i < n ; i++) {
        cout.width(7);
        cout << array[i];
    }
    cout << endl;
}

int main () {
    int table1[] = {-3, -41, -5, -7, 9, -9, 11, -13, 17};
    const int num = sizeof(table1)/sizeof(int);

    cout << "unsorted" << endl;
    display(table1,num);

    // variation 1: create explicitly a compare function and pass it to
    // the bubble_sort function as a parameter

```

```

less<int> lessCompareFunction;
bubble_sort(table1,num,lessCompareFunction);

cout << "sorted with lessCompareFunction" << endl;
display(table1,num);

// variation 2: don't create explicitly a compare function and
// call the bubble_sort function directly
bubble_sort(table1,num,less<int>());
// the same as bubble_sort(table1,num,less<int>.operator());

cout << "sorted with less<int>()" << endl;
display(table1,num);

// variation 3: call user comparison function, if wanted
bubble_sort(table1,num,absolute_less<int>());

cout << "sorted with absolute_less<int>()" << endl;
display(table1,num);
return 0;
}

```

Saída:

```

unsorted
  -3   -41   -5   -7    9   -9   11  -13   17
sorted with lessCompareFunction
  17   11    9   -3   -5   -7   -9  -13  -41
sorted with less<int>()
  17   11    9   -3   -5   -7   -9  -13  -41
sorted with absolute_less<int>()
 -41   17  -13   11    9   -9   -7   -5   -3

```

16.2.4 Classes aritméticas e lógicas

De forma muito semelhante às classes de comparação, STL também possui classes para funções aritméticas e lógicas. A tabela abaixo ilustra-as.

Definição do Objeto	Chamada	Retorno
<code>plus<T> X;</code>	<code>X(x,y)</code>	<code>x + y</code>
<code>minus<T> X;</code>	<code>X(x,y)</code>	<code>x - y</code>
<code>multiplies<T> X;</code>	<code>X(x,y)</code>	<code>x * y</code>
<code>divides<T> X;</code>	<code>X(x,y)</code>	<code>x / y</code>
<code>modulus<T> X;</code>	<code>X(x,y)</code>	<code>x % y</code>
<code>negate<T> X;</code>	<code>X(x)</code>	<code>-x</code>
<code>logical_and<T> X;</code>	<code>X(x,y)</code>	<code>x && y</code>
<code>logical_or<T> X;</code>	<code>X(x,y)</code>	<code>x y</code>
<code>logical_not<T> X;</code>	<code>X(x)</code>	<code>!x</code>

Classes de aritméticas e lógicas do STL

16.2.5 Complexidade de um algoritmo

É importante que se tenha uma noção de como medir a eficiência de um algoritmo (por exemplo um algoritmo de busca). Para isso uma das notações mais usadas é a notação O , que dá a ordem de grandeza de como cresce o tempo máximo de processamento do algoritmo em relação ao número de elementos contidos. As classificações típicas que se pode definir com a notação O são: $O(1)$, $O(n)$, $O(n^2)$, $O(\log n)$, $O(n \log n)$, etc.

Definição da ordem de complexidade O :

Seja $f(n)$ a função que retorna o tempo de execução de um algoritmo que contém n elementos. Esse algoritmo possui complexidade $O(g(n))$ se e somente se existem as constantes positivas c e n_0 tais que $|f(n)| \leq c|g(n)|$ é verdadeiro para qualquer $n \geq n_0$.

Exemplos:

Algoritmo	Frequência	complexidade (de tempo)
$x = x + y$	1	constante
for i=1 to n do $x = x + y$ end do	n	linear
for i=1 to n do for j=1 to n $x = x + y$ end do end do	n^2	quadrático
n = (inteiro) k = 0 while n > 0 do $n = n / 2$ $x = x + y$ end do	$\log n$	logaritmico

16.2.5.1 Algumas regras

Regra	Exemplo
$O(\text{const} * f) = O(f)$	$O(2n) = O(n)$
$O(f * g) = O(f) * O(g)$	$O((22n) * n) = O(22n) * O(n) = O(n) * O(n) = O(n^2)$
$O(f / g) = O(f) / O(g)$	$O((4n^3) / n) = O(4n^2) = O(n^2)$
$O(f + g) = \text{dominante}(O(f), O(g))$	$O(n^5 + n^3) = O(n^5)$

Exemplos:

1. Algoritmo tipo “busca linear”

Seja um problema como esse. Há n fichas telefônicas desordenadas. O trabalho é encontrar uma ficha que de um nome de entrada. A busca é linear. Como a localização no nome é aleatória nas fichas, na média pode-se dizer o tempo de resposta é o tempo de procurar $n/2$ fichas. Portanto a ordem de complexidade é $O(n/2) = O(n)$.

2. Algoritmo tipo “busca binária”

Vejamos agora como fica o algoritmo anterior no caso de fichas ordenadas. A partir de um nome de entrada, procura-se no meio das fichas. A cada inferência, ou encontra-se a ficha, ou reduz-se o espaço de procura pela metade. Não é difícil provar que nesse caso a ordem de complexidade é $O(\log n)$.

16.3 Iterador (iterator)

Iteradores são usados por algoritmos para referenciar objetos dentro de contenedores. O iterador mais simples são ponteiros. Essa seção descreve outros tipos de iteradores e suas propriedades em geral. Os iteradores funcionam em parceria muito próxima com contenedores, que são explicados em maior detalhe na próxima seção.

As propriedades essenciais dos iteradores, como mostrado na seção 16.2 são avançar (++), referenciar (*) e comparação (!= e ==). Se o iterador não é um ponteiro comum, mas um objeto de uma classe de iterador, essas propriedades são implementadas por meio de funções de operação.

16.3.1 Iteradores padrão para inserção em contenedores

O programa abaixo ilustra o funcionamento de `back_insert_iterator` e `front_insert_iterator`, que são usados para criar iteradores para o objeto contenedor `myListObject`, que por sua vez foi criado a partir da classe genérica STL `list`, aplicando-se a ela o tipo `double`.

Para facilitar a exibição do objeto contenedor, foi criada a função global genérica (com template) chamada `VBShowContainer`.

O programa cria uma lista vazia inicial, com 2 zeros. São criados iteradores para inserção no fim (`myBackInsertIterator`) e no início (`myFrontInsertIterator`). Em seguida, o programa insere elementos no início da lista, e posteriormente insere elementos no final da lista. Em todos os casos, o conteúdo da lista é mostrado, para que se acompanhe o que está acontecendo com a lista. Preste atenção na ordem com que os elementos são inseridos.

```
#include <iostream>
#include <list> // STL, list
#include <iterator> // STL, back_insert_iterator
using namespace std;
#define TYPE double
typedef list<TYPE> myListType;

template <class containerType>
void VBShowContainer(const containerType & containerObj) {
    typename containerType::const_iterator iteratorObj;
    if (containerObj.empty()) {
        cout << "====Container empty" << endl;
        return;
    }
    cout << "====show the container contents, size="
```

```

        << containerObj.size() << endl;
    for (iteratorObj = containerObj.begin() ;
        iteratorObj != containerObj.end() ;
        iteratorObj++)
        cout << *iteratorObj << endl;
};

int main () {
    int i;

    myListType myListObject(2); // 2 zeros

    // create user defined iterators
    back_insert_iterator<myListType> myBackInsertIterator(myListObject);
    front_insert_iterator<myListType> myFrontInsertIterator(myListObject);

    VBShowContainer(myListObject);

    // insertion by means of operations *, ++, =
    for (i = 1 ; i < 3 ; i++)
        *myBackInsertIterator++ = i;

    VBShowContainer(myListObject);

    // insertion by means of operations *, ++, =
    for (i = 10 ; i < 13 ; i++)
        *myFrontInsertIterator++ = i;

    VBShowContainer(myListObject);
    return 0;
}

```

Saída do programa.

```

=====show the container contents, size=2
0
0
=====show the container contents, size=4
0
0
1
2
=====show the container contents, size=7
12
11
10
0
0
1
2

```

Uma forma alternativa de se criar um iterador tipo “back” e tipo “front” é usar o `insert_iterator`. Por exemplo: as linhas abaixo poderiam ter sido substituídas

```

back_insert_iterator<myListType> myBackInsertIterator(myListObject);
front_insert_iterator<myListType> myFrontInsertIterator(myListObject);

```

por essas linhas

```

insert_iterator<myListType> myBackInsertIterator(myListObject,myListObject.end());
insert_iterator<myListType> myFrontInsertIterator(myListObject,myListObject.begin());

```

16.4 Contenedor (container)

Em resumo pode-se dizer que um contenedor é um objeto que contém outros objetos. Os contenedores mais importantes do STL estão mostrados abaixo.

16.4.1 Vector

O vetor é um contenedor que pode ser referenciado diretamente com `operator[]`. Além disso, pode-se acrescentar dados ao vetor com `back_insert_iterator` (mas não com `front_insert_iterator`). Como todo contenedor, possui métodos `size`, `begin`, e `end`.

```
#include <iostream>
#include <vector> // STL, vector
#include <iterator> // STL, back_insert_iterator

// the same VBShowContainer global function as above

using namespace std;

#define TYPE double
typedef vector<TYPE> myVectorType;

int main () {
    myVectorType myVectorObject(3); // the container object. 5 zeros
    myVectorType::iterator myVectorIterator;

    VBShowContainer(myVectorObject);

    // create user defined iterators
    back_insert_iterator<myVectorType> myBackInsertIterator(myVectorObject);

    // insertion by means of operations *, ++, =
    for (int i = 10 ; i < 13 ; i++)
        *myBackInsertIterator++ = i;

    VBShowContainer(myVectorObject);

    // referencing directly the vector with operator[]
    cout << "----- Referencing directly" << endl;
    myVectorObject[1] = 55; // load some data directly
    int size = myVectorObject.size();
    for (i = 0 ; i < size ; i++)
        cout << myVectorObject[i] << endl;
    return 0;
}
```

Saída do programa.

```
=====show the container contents, size=3
0
0
0
=====show the container contents, size=6
0
0
0
10
11
12
```

```

----- Referencing directly
0
55
0
10
11
12

```

16.4.2 List

No exemplo abaixo, os métodos `push_front`, `push_back`, `assign`, `empty` e `erase` são mostrados.

Um outro exemplo de uso de `list` foi mostrado na seção 16.2.5.

```

#include <list>
#include <iostream>
// the same VBShowContainer global function as above
using namespace std ;
typedef list<int> myListType;
int main() {
    myListType listOne;
    myListType listAnother;

    // Add some data to list one
    listOne.push_front (2);
    listOne.push_front (1);
    listOne.push_back (3);
    VBShowContainer(listOne); // see list one

    // Add some data to list another
    listAnother.push_front(4);
    listAnother.assign(listOne.begin(), listOne.end());

    VBShowContainer(listAnother); // see list another
    listAnother.assign(4, 1);
    VBShowContainer(listAnother); // see list another
    listAnother.erase(listAnother.begin());
    VBShowContainer(listAnother); // see list another
    listAnother.erase(listAnother.begin(), listAnother.end());
    VBShowContainer(listAnother); // see list another
    return 0;
}

```

Saída do programa:

```

=====show the container contents, size=3
1
2
3
=====show the container contents, size=3
1
2
3
=====show the container contents, size=4
1
1
1
1
=====show the container contents, size=3
1
1
1

```


=====`Container empty`

16.4.3 Pilha (Stack)

Uma pilha (stack) é um elemento contenedor em que “o primeiro que entra, é o último que sai”. Para colocar dados numa pilha, usa-se o método `push`. Somente pode-se acessar o último objeto que entrou na pilha. O acesso a esse objeto é feito pelo método `top` (tanto para ler quanto para escrever). Para jogar fora quem está no topo e pegar o que está em baixo, há o método `pop`. Há ainda o método `empty`, que verifica se a pilha está vazia ou não.

```
#include <stack>
#include <iostream>
using namespace std ;
#define TYPE float
typedef stack<TYPE> myStackType;
int main() {
    myStackType stackObject;
    cout << "stackObject is " <<
        (stackObject.empty() ? "empty": "full") << endl;

    stackObject.push(2); // push some data
    stackObject.push(4); // push some data
    stackObject.push(6); // push some data
    stackObject.push(8); // push some data

    cout << "stackObject is " <<
        (stackObject.empty() ? "empty": "full") << endl;

    // Modify the top item.
    if (!stackObject.empty()) {
        stackObject.top()=22;
    }

    // Repeat until stack is empty
    while (!stackObject.empty()) {
        const TYPE & t=stackObject.top();
        cout << "stack object = " << t << endl;
        stackObject.pop();
    }
    return 0;
}
```

Saída do programa

```
stackObject is empty
stackObject is full
stack object = 22
stack object = 6
stack object = 4
stack object = 2
```

16.4.4 Fila (Queue)

Uma fila (queue) permite que se insira dados por uma ponta e retire esse dado pela outra ponta.

A interface da fila contém os métodos abaixo:

- `bool empty();` retorna true se a fila está vazia.

- `size_type size() const`; retorna o tamanho da fila.
- `value_type & front()`; retorna o valor no início da fila.
- `const value_type & front() const`; retorna o valor no início da fila.
- `value_type & back()`; retorna o valor no fim da fila.
- `const value_type & back() const`; retorna o valor no fim da fila.
- `void push(const value_type & x)`; acrescenta o objeto x a fila.
- `void pop()`; apaga o primeiro objeto da fila.

```
#include <iostream>
#include <queue>
using namespace std ;
#define TYPE int
typedef queue<TYPE> queueIntType;
typedef queue<char*> queueCharType;
int main()
{
    int size;
    queueIntType myIntQueueObject;
    queueCharType myCharQueueObject;

    // Insert items in the queue(uses list)
    myIntQueueObject.push(1);
    myIntQueueObject.push(2);
    myIntQueueObject.push(3);
    myIntQueueObject.push(4);

    // Output the size of queue
    size = myIntQueueObject.size();
    cout << "size of int queue is:" << size << endl;

    // Output items in queue using front(), and delete using pop()
    while (!myIntQueueObject.empty()) {
        cout << myIntQueueObject.front() << endl; // read object from queue
        myIntQueueObject.pop(); // delete object from queue
    }

    // Insert items in the char queue
    myCharQueueObject.push("Maria");
    myCharQueueObject.push("Alan");
    myCharQueueObject.push("Sergio");
    myCharQueueObject.push("Marcia");
    myCharQueueObject.push("Paulo");

    // Output the item inserted last using back()
    cout << "The last element of char* queue is " <<
        myCharQueueObject.back() << endl;

    // Output the size of queue
    size = myCharQueueObject.size();
    cout << "size of char* queue is:" << size << endl;

    // Output items in queue using front(), and delete using pop()
    while (!myCharQueueObject.empty()) {
        cout << myCharQueueObject.front() << endl; // read object from queue
        myCharQueueObject.pop(); // delete object from queue
    }
}
```

```

    }
    return 0;
}

```

Saída do programa:

```

size of int queue is:4
1
2
3
4
The last element of char* queue is Paulo
size of char* queue is:5
Maria
Alan
Sergio
Marcia
Paulo

```

16.4.5 Fila com prioridade (priority queue)

A fila com prioridade é diferente da fila comum por permitir atribuir prioridades aos objetos da fila. Quando alguém é retirado da fila, é respeitada a ordem de prioridades estabelecida.

No programa abaixo, foi criada uma classe `myClass`, que armazena o nome e idade de uma pessoa. Para essa classe, definiu-se o `operator<` e `operator>` baseados na observação dos campos de idade apenas (esse será o critério de prioridade). Definiu-se também nessa classe o operador `insersor` (`operator<<`) para permitir fácil exibição no console.

Em seguida, criou-se um `queueObject` a partir do aparentemente estranho tipo ressaltado na linha abaixo.

```
priority_queue<myClass , vector<myClass> , greater<myClass> >
```

Trata-se de uma classe definida em template, sendo que o primeiro tipo é a classe a ser armazenada, o segundo tipo é uma classe contenedora e o terceiro tipo é uma classe de comparação STL (veja tabela na página 369). No caso, escolheu-se a classe `greater<T>` para comparação, que é baseada no `operator>` da classe em questão.

```

#include <iostream>
#include <queue>
using namespace std ;

class myClass {
public:
    myClass(char *name, int age) { m_name = name; m_age = age; }
    char *m_name;
    int m_age;
    friend bool operator>(myClass x, myClass y) { return !(x.m_age > y.m_age); }
    friend bool operator<(myClass x, myClass y) { return !operator>(x,y); }
    friend ostream & operator<< (ostream & stream , const myClass & obj ) {
        stream << "Name:" << obj.m_name
                << "\tAge:" << obj.m_age;
        return stream;
    }
};

```

```

int main () {
    priority_queue<myClass , vector<myClass> , greater<myClass> > queueObject;
    queueObject.push(myClass("Sergio", 36));
    queueObject.push(myClass("Alan", 26));
    queueObject.push(myClass("Marcia", 22));
    queueObject.push(myClass("Paulo", 28));
    queueObject.push(myClass("Ana", 21));

    // Output the size of queue
    int size = queueObject.size();
    cout << "size of int queue is:" << size << endl;

    // Output items in queue using front(), and delete using pop()
    while (!queueObject.empty()) {
        cout << queueObject.top() << endl; // read object from queue
        queueObject.pop(); // delete object from queue
    }

    return 0;
}

```

A saída do programa é mostrada abaixo. Repare que os objetos entraram desordenados na fila, e saíram ordenados de acordo com a prioridade escolhida (no caso sai primeiro quem tem idade menor).

```

Name:Ana      Age:21
Name:Marcia   Age:22
Name:Alan     Age:26
Name:Paulo    Age:28
Name:Sergio   Age:36

```

Caso se deseje mudar a prioridade para sair primeiro que tem idade *maior*, basta mudar a classe de comparação de `greater<T>` para `less<T>`. Se isso for feito, a saída do programa fica como mostrado abaixo.

```

size of int queue is:5
Name:Sergio   Age:36
Name:Paulo    Age:28
Name:Alan     Age:26
Name:Marcia   Age:22
Name:Ana      Age:21

```

16.4.6 Contenedores associativos ordenados

São contenedores que permitem acesso rápido aos dados através de chaves que não necessariamente coincidem com os dados propriamente ditos, isto é, podem ser chaves estrangeiras.

O STL armazena as chaves de forma ordenada, apesar de isso não ser requisito para o funcionamento das tarefas. Essa característica é apenas um detalhe de implementação, que balanceia o armazenamento dos objetos (chaves). Devido ao ordenamento, o acesso aos elementos é muito rápida. O contenedor das chaves é geralmente uma árvore (tree), que fica balanceada devido ao ordenamento. Uma alternativa para tornar o acesso ainda mais rápido é usar um

hash³¹ para armazenar as chaves. Caso o hash seja usado, o uso de memória é maior, mas a ordem de acesso é $O(1)$, ao invés de $O(\log N)$ que ocorre no caso de se usar árvore.

Em set e multiset, os dados eles mesmo são usados como chaves. Em map e multimap, as chaves e os dados são diferentes. Os 4 tipos de contenedores associativos ordenados estão mostrados abaixo.

16.4.6.1 Set

No set, as chaves (key) coincidem com os dados. Não pode haver elementos com a mesma chave.

No programa abaixo, um setObject é definido, e alguns dados são acrescentados. A função global myCheck testa se o objeto existe. Para isso, é chamado o método find, que retorna o iterador apontando para o objeto que foi encontrado, ou apontando para o fim caso o objeto não tenha sido encontrado. A variável booleana exist armazena a informação de existência ou não do objeto procurado. A função global myShowExist retorna char* informando “existe” ou “não existe” de acordo com o argumento recebido.

Repare que o mesmo dado (12) é inserido duas vezes. Na segunda vez que o dado é inserido, ele é ignorado.

Esse programa gera alguns warnings quando programado em Visual C++. Esses warnings podem ser ignorados. Em g++ não é gerado qualquer warning.

```
#include <iostream>
#include <set>
using namespace std;

// the same VBShowContainer global function as above

#define TYPE int
typedef set<TYPE> setType;

char *myShowExist(bool b) {
    if (b)
        return " exists";
    else
        return " does not exist";
}

// argument can not be "const setType & setObject"
void myCheck(setType & setObject, const TYPE & k) {
    setType::iterator it; // iterator
    it = setObject.find(k);
    bool exist = it!=setObject.end();
    cout << "Item " << k << myShowExist(exist) << endl;
}
```

³¹ hash significa “carne picada”, ou “picar”. Não creio que sejam termos razoáveis para serem usar em português, portanto o termo usado será em inglês mesmo.

```

int main() {
    setType setObject;    // container object
    setType::iterator it;  // iterator

    // fill with some data
    setObject.insert(5);
    setObject.insert(8);
    setObject.insert(12);
    setObject.insert(12); // insert same data twice. Ignored
    setObject.insert(7);
    VBShowContainer(setObject);

    // erasing an object
    int i=7;
    it = setObject.find(i);
    setObject.erase(it);

    // check if data exists in the container object
    myCheck(setObject,8);
    myCheck(setObject,6);
    return 0;
}

```

Saída do programa.

```

=====show the container contents, size=3
5
8
12
Item 8 exists
Item 6 does not exist

```

Um outro exemplo interessante usando set é mostrado abaixo. Repare que um array de dados é passado para o construtor do setObject, e com isso, todos os dados entram no objeto. Como deveria ser, os dados repetidos não entram.

```

#include <set>
#include <iostream>
// the same VBShowContainer global function as above
#define TYPE double
int main () {
    TYPE array[] = { 1.0, 2.0, 2.0, 5.0, 6.5, 6.8, 7.0, 1.0, 2.2 };
    int count = sizeof(array) / sizeof(TYPE);
    set<TYPE> setObject(array, array + count);
    VBShowContainer(setObject);
    return 0;
}

```

Saída do programa.

```

=====show the container contents, size=7
1
2
2.2
5
6.5
6.8
7

```

Exercício:

Faça um programa que leia um arquivo de texto, em que cada linha contém um email. Mas é sabido que há emails repetidos na lista. O programa deve abrir esse

arquivo como leitura e salvar outro arquivo com o mesmo conteúdo, apenas eliminando os emails repetidos.

16.4.6.2 Multiset

O multiset é parecido com o set, exceto que aqui é permitido dados com mesma chave. O programa abaixo é uma repetição do último programa com set, apenas mudando para multiset. A diferença é que os elementos repetidos são permitidos.

```
#include <set>
#include <iostream>
// the same VBShowContainer global function as above
#define TYPE double
int main()
{
    TYPE array[] = { 1.0, 2.0, 2.0, 5.0, 6.5, 6.8, 7.0, 1.0, 2.2 };
    int count = sizeof(array) / sizeof(TYPE);
    multiset<TYPE> setObject(array, array + count);
    VBShowContainer(setObject);
    return 0;
}
```

Saída do programa.

```
=====show the container contents, size=9
1
1
2
2
2.2
5
6.5
6.8
7
```

16.4.6.3 Map

O map é parecido com o set, exceto que os dados e a chave são distintos. No map, não é permitido dados com mesma chave. O map é um contenedor de um par (veja 16.2.1.1) de elementos.

No programa de exemplo abaixo, um objeto map contém os elementos (e os mapeamentos) de long e string. É criado um tipo para armazenar o mapa e um tipo que armazena valor, no caso `valuePairType`. O programa cria o objeto map, carrega-o com dados pelo código fonte. Para exemplificar, um dos dados tem a mesma chave, portanto não será efetivamente incluído. Em seguida, um laço mostra o conteúdo total do map.

A próxima parte do programa pede ao usuário para entrar uma chave pelo console (no caso um long). Com essa chave, busca-se no map o elemento (par) que corresponde com o método `find`. O algoritmo de busca é feito com ordem $O(\log N)$. Depois que o iterador é encontrado, o acesso ao elemento é feito com $O(1)$. Outra possibilidade é usar o operador `[],` que retorna o elemento `second` do par com a mesma ordem $O(\log N)$.

```

#include <map>
#include <string>
#include <iostream>
using namespace std;

typedef map<long, string> mapType;
typedef mapType::value_type valuePairType;

int main()
{
    mapType mapObject;
    mapType::iterator it;    // iterator

    // add some data.
    mapObject.insert(valuePairType(1234567,"Sergio"));
    mapObject.insert(valuePairType(232134,"Marcos"));
    mapObject.insert(valuePairType(3423423,"Ana"));
    mapObject.insert(valuePairType(938423,"Marcia"));
    mapObject.insert(valuePairType(23434534,"Mateus"));
    mapObject.insert(valuePairType(746545,"Camila"));

    // Attention: Sergio and Gabriel have the same key !
    // So, Gabriel is not input !
    mapObject.insert(valuePairType(1234567,"Gabriel"));

    int size = mapObject.size();
    cout << "==== map size = " << size << endl;
    // loop to show the map
    it = mapObject.begin();
    while (it != mapObject.end()) {
        cout << it->first << " \t: "    // number
              << it->second           // name
              << endl;
        it++;
    }

    cout << "Enter a number:";
    long number;
    cin >> number;

    it = mapObject.find(number);    // O(logN)

    if (it != mapObject.end()) {
        cout << "Number found" << endl;
        cout << "Correspondent name is: "
              << it->second << endl    // O(1)
              << "again: " << mapObject[number] << endl; // O(logN)
    }
    else
        cout << "Number not found" << endl;
    return 0;
}

```

Saída do programa.

```

==== map size = 6
232134      : Marcos
746545      : Camila
938423      : Marcia
1234567     : Sergio
3423423     : Ana
23434534    : Mateus
Enter a number:23434534

```



```
Number found
Correspondent name is: Mateus
again: Mateus
```

16.4.6.4 *Multimap*

O multimap é parecido com o map, exceto que é permitido dados com mesma chave.

16.5 Algoritmos

16.5.1 `remove_if`

O algoritmo `remove_if()` é parecido com o algoritmo `remove`. Os dois primeiros argumentos marcam o início e o final da sequência. O terceiro argumento é o predicado. Diferentemente de `remove()`, o algoritmo `remove_if()` usa o predicado dado para selecionar os elementos que removerá. Com isso, o usuário do algoritmo poderá introduzir uma regra para a remoção, de acordo com a sua própria lógica. O uso do `remove_if()` é similar ao `remove()`, sendo a única diferença a necessidade de se definir o predicado.

No exemplo abaixo, o algoritmo `remove_if()` é usado para remover vogais de um objeto do tipo string. O predicado deve identificar (retornar booleano) se uma letra é vogal ou não.

Lembre-se que o `remove_if()` realmente não apaga os elementos da string. Essa função simplesmente move os elementos para o fim e retorna um iterator para a posição. Utilizando-se desse iterator uma nova string, que não contém vogais, é criada.

```
#include <string>
#include <iostream>
#include <functional> // unary_function
#include <algorithm> // for remove_if
using namespace std;

/* prototype of remove_if
template <class ForwardIterator, class T>
ForwardIterator remove_if (ForwardIterator first,
                          ForwardIterator last,
                          Predicate pred);

*/

template <class T>
class is_vowel: public unary_function<T,T>
{
public:
    bool operator()(T t) const {
        if ((t=='a')||(t=='e')||(t=='i')||(t=='o')||(t=='u'))
            return true; //t is a vowel
        return false; // t is not a vowel
    }
};

int main()
```

```

{
    string original;
    original = "abcdefghijklmnopqrstuvwxyz";

    // Next, we call remove_if() to remove all the vowels from the input string.
    // We create a temporary object of the specialization is_vowel and pass
    // it as the predicate of remove_if():
    // move vowels to end and store the position where the vowels start
    string::iterator it= remove_if(original.begin(),
                                   original.end(),
                                   is_vowel<char>());

    // create new string from original using previous iterator
    string no_vowels(original.begin(),it);
    cout << no_vowels << endl;
    return 0;
}

```

A saída do programa é como mostrado abaixo.

```
bcdfghjklmnpqrstvwxyz
```


17 Componentes de Programação

Nesse capítulo são mostrados alguns componentes de programação com utilidades diversas. Alguns são para Windows & DOS, outros são para unix (testado em Linux).

17.1 Para Windows & DOS

Programas que usem diretamente o barramento de entrada e saída da CPU, o que equivale a instruções de in e out em assembly, não funcionam em Windows NT (nem em Windows 2000 que é baseado em tecnologia NT). Mas funciona normalmente em DOS em Windows 9x (que é baseado em tecnologia DOS).

17.1.1 Programa para listar o diretório corrente (Visual C++)

```
#include <windows.h>
#include <iostream.h>    // cout

int main () {
    WIN32_FIND_DATA f;
    HANDLE h;
    bool done;

    h = FindFirstFile("*.*", &f);
    done = (h == 0);
    while (!done) {
        DWORD attribs = GetFileAttributes(file);
        bool isDir = (attribs & FILE_ATTRIBUTE_DIRECTORY) != 0;
        if (isDir) cout << "DIR ";
        cout << f.cFileName << endl;
        done = !FindNextFile(h, &f);
    }
    return 0;
}
```

Resultado

```
.
..
card_1cpp.cpp
teste6.dsp
teste6.plg
cardcpp_2.cpp
```

Um código alternativo, para listar arquivos, eliminando-se o arquivo “.”, “..” e os arquivos que são diretórios.

```
#include <windows.h>
#include <iostream.h>    // cout
#include "vblib.h"

int main ()
{
    WIN32_FIND_DATA f;
    HANDLE h;
```

```

bool done;

h = FindFirstFile("*.*", &f);
done = (h == 0);
VBString file;
while (!done)
{
    file = f.cFileName;
    bool isDirectory = (file == VBString(".") || file == VBString(".."));
    if (!isDirectory)
        cout << file << endl;
    done = !FindNextFile(h, &f);
}
return 0;
}

```

Resultado

```

card_1cpp.cpp
teste6.dsp
teste6.plg
cardcpp_2.cpp

```

17.1.2 Porta Serial

A classe VBSerial, mostrada abaixo com os arquivos VBSerial.h e VBSerial.cpp pode ser usada para controlar a porta serial em Windows. Essa classe faz uma pequena manipulação de strings, a partir de VBString. Portanto, é preciso incluir VBLib.cpp no projeto, e VBLib.h no path dos includes para usar essa classe.

```

// VBSerial.h

#ifndef __VB_SERIAL_H__
#define __VB_SERIAL_H__

#include "vblib.h"

#define FC_DTRDSR      0x01
#define FC_RTSCTS      0x02
#define FC_XONXOFF     0x04
#define ASCII_BEL      0x07
#define ASCII_BS       0x08
#define ASCII_LF       0x0A
#define ASCII_CR       0x0D
#define ASCII_XON      0x11
#define ASCII_XOFF     0x13

class VBSerial {
public:
    VBSerial();
    ~VBSerial();
    BOOL Open( int nPort = 2, int nBaud = 9600 );
    BOOL Close( void );
    int ReadData( void *, int );
    int SendData( const char *, int );
    int ReadDataWaiting( void );
    BOOL IsOpened( void ){ return( m_bOpened ); }
protected:
    BOOL WriteCommByte( unsigned char );
    HANDLE m_hIDComDev;

```

```

        OVERLAPPED m_OverlappedRead, m_OverlappedWrite;
        BOOL m_bOpened;
};

#endif

// VBSerial.cpp

#include "stdafx.h"
#include "VBSerial.h"

VBSerial::VBSerial()
{
    memset( &m_OverlappedRead, 0, sizeof( OVERLAPPED ) );
    memset( &m_OverlappedWrite, 0, sizeof( OVERLAPPED ) );
    m_hIDComDev = NULL;
    m_bOpened = FALSE;
}

VBSerial::~VBSerial()
{
    Close();
}

BOOL VBSerial::Open( int nPort, int nBaud )
{
    if( m_bOpened ) return( TRUE );
    VBString szPort;
    VBString szComParams;
    DCB dcb;
    szPort = "COM";
    szPort += nPort;
    m_hIDComDev = CreateFile( szPort, GENERIC_READ | GENERIC_WRITE, 0, NULL,
        OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED, NULL );
    if( m_hIDComDev == NULL ) return( FALSE );
    memset( &m_OverlappedRead, 0, sizeof( OVERLAPPED ) );
    memset( &m_OverlappedWrite, 0, sizeof( OVERLAPPED ) );
    COMMTIMEOUTS CommTimeOuts;
    CommTimeOuts.ReadIntervalTimeout = 0xFFFFFFFF;
    CommTimeOuts.ReadTotalTimeoutMultiplier = 0;
    CommTimeOuts.ReadTotalTimeoutConstant = 0;
    CommTimeOuts.WriteTotalTimeoutMultiplier = 0;
    CommTimeOuts.WriteTotalTimeoutConstant = 5000;
    SetCommTimeouts( m_hIDComDev, &CommTimeOuts );

    szComParams = "COM";
    szComParams += nPort;
    szComParams += ":";
    szComParams += nBaud;
    szComParams += ",n,8,1";

    m_OverlappedRead.hEvent = CreateEvent( NULL, TRUE, FALSE, NULL );
    m_OverlappedWrite.hEvent = CreateEvent( NULL, TRUE, FALSE, NULL );
    dcb.DCBlength = sizeof( DCB );
    GetCommState( m_hIDComDev, &dcb );
    dcb.BaudRate = nBaud;
    dcb.ByteSize = 8;
    unsigned char ucSet;

```

```

ucSet = (unsigned char) ( ( FC_RTSCST & FC_DTRDSR ) != 0 );
ucSet = (unsigned char) ( ( FC_RTSCST & FC_RTSCST ) != 0 );
ucSet = (unsigned char) ( ( FC_RTSCST & FC_XONXOFF ) != 0 );
if( !SetCommState( m_hIDComDev, &dcb ) ||
    !SetupComm( m_hIDComDev, 10000, 10000 ) ||
    m_OverlappedRead.hEvent == NULL ||
    m_OverlappedWrite.hEvent == NULL ) {
    DWORD dwError = GetLastError();
    if( m_OverlappedRead.hEvent != NULL ) CloseHandle( m_OverlappedRead.hEvent );
    if( m_OverlappedWrite.hEvent != NULL )
        CloseHandle( m_OverlappedWrite.hEvent );
    CloseHandle( m_hIDComDev );
    return( FALSE );
}
m_bOpened = TRUE;
return( m_bOpened );
}

BOOL VBSerial::Close( void )
{
    if( !m_bOpened || m_hIDComDev == NULL ) return( TRUE );
    if( m_OverlappedRead.hEvent != NULL ) CloseHandle( m_OverlappedRead.hEvent );
    if( m_OverlappedWrite.hEvent != NULL ) CloseHandle( m_OverlappedWrite.hEvent );
    CloseHandle( m_hIDComDev );
    m_bOpened = FALSE;
    m_hIDComDev = NULL;
    return( TRUE );
}

BOOL VBSerial::WriteCommByte( unsigned char ucByte )
{
    BOOL bWriteStat;
    DWORD dwBytesWritten;

    bWriteStat = WriteFile( m_hIDComDev, (LPSTR) &ucByte, 1,
        &dwBytesWritten, &m_OverlappedWrite );
    if( !bWriteStat && ( GetLastError() == ERROR_IO_PENDING ) ) {
        if( WaitForSingleObject( m_OverlappedWrite.hEvent, 1000 ) )
            dwBytesWritten = 0;
        else {
            GetOverlappedResult( m_hIDComDev, &m_OverlappedWrite,
                &dwBytesWritten, FALSE );
            m_OverlappedWrite.Offset += dwBytesWritten;
        }
    }
    return( TRUE );
}

int VBSerial::SendData( const char *buffer, int size )
{
    if( !m_bOpened || m_hIDComDev == NULL ) return( 0 );
    DWORD dwBytesWritten = 0;
    int i;
    for( i=0; i<size; i++ ) {
        WriteCommByte( buffer[i] );
        dwBytesWritten++;
    }
    return( (int) dwBytesWritten );
}

int VBSerial::ReadDataWaiting( void )
{

```

```

    if( !m_bOpened || m_hIDComDev == NULL ) return( 0 );
    DWORD dwErrorFlags;
    COMSTAT ComStat;
    ClearCommError( m_hIDComDev, &dwErrorFlags, &ComStat );
    return( (int) ComStat.cbInQue );
}

int VBSerial::ReadData( void *buffer, int limit )
{
    if( !m_bOpened || m_hIDComDev == NULL ) return( 0 );
    BOOL bReadStatus;
    DWORD dwBytesRead, dwErrorFlags;
    COMSTAT ComStat;
    ClearCommError( m_hIDComDev, &dwErrorFlags, &ComStat );
    if( !ComStat.cbInQue ) return( 0 );
    dwBytesRead = (DWORD) ComStat.cbInQue;
    if( limit < (int) dwBytesRead ) dwBytesRead = (DWORD) limit;
    bReadStatus = ReadFile( m_hIDComDev, buffer, dwBytesRead,
        &dwBytesRead, &m_OverlappedRead );
    if( !bReadStatus ) {
        if( GetLastError() == ERROR_IO_PENDING ) {
            WaitForSingleObject( m_OverlappedRead.hEvent, 2000 );
            return( (int) dwBytesRead );
        }
        return( 0 );
    }
    return( (int) dwBytesRead );
}

```

17.1.3 Porta Paralela

A pinagem da porta paralela pode ser vista na tabela abaixo.

Descrição dos pinos			
Pino	Sinal (* significa lógica inversa)	Direção	Descrição
1	STROBE*	OUT	sinal de Controle
2	DATA 0	OUT	bit de dado D0
3	DATA 1	OUT	bit de dado D1
4	DATA 2	OUT	bit de dado D2
5	DATA 3	OUT	bit de dado D3
6	DATA 4	OUT	bit de dado D4
7	DATA 5	OUT	bit de dado D5
8	DATA 6	OUT	bit de dado D6
9	DATA 7	OUT	bit de dado D7
10	ACKNOWLEDGE*	IN	sinal de controle
11	BUSY	IN	sinal de controle
12	PAPER END	IN	sinal de controle
13	SELECT OUT	IN	sinal de

			controle
14	AUTO FEED*	OUT	sinal de controle
15	ERROR*	IN	sinal de controle
16	INITIALIZE PRINTER	OUT	sinal de controle
17	SELECT INPUT*	OUT	sinal de controle
18	GROUND	-	Terra (ground)
19	GROUND	-	Terra (ground)
20	GROUND	-	Terra (ground)
21	GROUND	-	Terra (ground)
22	GROUND	-	Terra (ground)
23	GROUND	-	Terra (ground)
24	GROUND	-	Terra (ground)
25	GROUND	-	Terra (ground)

O programa abaixo joga constantes byte na porta paralela usando diretamente a instrução outportb.

```
#include <iostream.h>
#include <conio.h> // _outp
#define LPT1_DATA 0x378

// the user out port function
void outportb(unsigned short port, unsigned short val) {
    // calls the outport function of the compiler
    _outp(port,val); // _outp is VisualC only
}

// the user in port function
int inportb(unsigned short port) {
    return (_inp(port)); // _inp is VisualC only
}

const unsigned char init_val = 128; // 1000 0000

int main () {
    unsigned char value = init_val;
    int i=0;
    while (1) {
        outportb(LPT1_DATA,value);
        cout << i++ << " ) press any key to for next byte, space to stop" << endl;
        char ch = getch();
        if (ch == ' ') break;
        value = value >> 1; // bitwise shift
        if (value==0)
            value = init_val;
    }
    return 0;
}
```

17.2 Componentes para unix (inclui Linux)

17.2.1 Programa para listar o diretório corrente

Para simplificar a listagem de arquivos no diretório corrente em unix, fez-se uma classe chamada VBFileDirectory. Essa classe é baseada nas funções lstat, S_ISDIR, opendir e readdir. Mas utilizar essas funções diretamente é um tanto trabalhoso, e o que a classe faz é colocar uma interface simples para a utilização dessas funções. Como se manipula strings no processo, utilizou-se a classe VBString para essa finalidade. Os diretórios “.” e “..” (diretório corrente e pai do corrente) são ignorados por conveniência, e esse trabalho também é feito pela classe.

```
// compile with "g++ thisFile.cpp vblib.cpp"
// because this source uses VBString (in vblib.cpp)
// vblib.h also required.

#include <sys/stat.h> // struct stat, S_ISDIR
#include <dirent.h> // DIR

#include "vblib.h"

class VBFileDirectory {
    DIR *m_dp;
    VBString m_path, m_dirPath;
    bool m_endOfList;
    bool m_isDir;

    bool p_goodName(VBString fileName) {
        return ((VBString (".") != fileName) && (VBString ("..") != fileName) );
    }
    void p_isDirectory (VBString file);
    VBString p_get();
public:
    VBFileDirectory () { // default constructor
        m_isDir = false;
    }

    bool isDir () {
        return m_isDir;
    }

    VBString getPath () {
        return m_dirPath;
    }

    VBString getNext() {
        return p_get();
    }

    VBString getFirst(const char *path) {
        m_path = path;
        m_dirPath = path;
        m_dp = opendir(path);
        m_endOfList = (m_dp == 0);
        if (m_endOfList) return "";
        return p_get();
    }
}
```

```

        bool eolist() {
            if (m_endOfList) closedir(m_dp);
            return m_endOfList;
        }
};

// set m_isDir true if argument is a directory
void VBFileDirectory::p_isDirectory (VBString file) {
    DIR *dp = opendir(file);
    struct stat statbuf;
    lstat(file,&statbuf);
    m_isDir = (S_ISDIR(statbuf.st_mode) != 0);
    if (m_isDir) {
        m_dirPath = file;
    }
}

VBString VBFileDirectory::p_get() {
    bool isGoodFileName;
    VBString ret;
    struct dirent *dirp;
    do {
        dirp = readdir(m_dp);
        m_endOfList = (dirp == 0);
        if (m_endOfList) return "";
        isGoodFileName = p_goodName(dirp->d_name);
    } while (!isGoodFileName);
    ret = dirp->d_name;
    ret = m_path + VBString("/") + ret;
    p_isDirectory(ret); // set m_isDir
    return ret;
}

```

A listagem de arquivos do diretório corrente é um algoritmo que deve ser recursivo, pois pode haver um número indeterminado de filhos de cada diretório. A recursividade é obtida com a função global myDir, que usa um objeto da classe VBFileDirectory. O primeiro diretório é obtido com o método getFirst, em que se passa o path de base. O laço de programação prossegue até que o método eolist (end-of-list) retorne verdadeiro. Enquanto isso não ocorrer, os próximos arquivos podem ser obtidos com o método getNext.

Para cada arquivo obtido, pode-se saber se é um diretório ou não com o método que retorna booleano chamado isDir. Caso o arquivo seja um diretório, opcionalmente pode-se acrescentar uma marcação adicional na tela, e também pode-se chamar a própria função myDir recursivamente. No caso de se chamar a função myDir recursivamente, termina-se por varrer toda a árvore de diretórios. Caso haja um diretório na árvore que é um link para o diretório base, o algoritmo entra num laço infinito.

```

void myDir (VBString path) {
    VBFileDirectory a;
    VBString str;
    for (str = a.getFirst(path) ; !a.eolist() ; str=a.getNext()) {
        // mark the ones that are directories
        if (a.isDir()) cout << "DIR ===== ";
        cout << str << endl;
    }
}

```

```

        if (a.isDir()) myDir(a.getPath()); // recursive call
    }
}

itn main ()
{
    const char * path = "/home/villas/test";
    cout << "base path = " << path << endl;
    myDir(path);
    return 0;
}

```

Resultado (no caso de um diretório de teste)

```

base path = /home/villas/test
/home/villas/test/text1.txt
DIR ===== /home/villas/test/dir_1
/home/villas/test/dir_1/text1.txt
/home/villas/test/dir_1/text2.txt
/home/villas/test/dir_1/text3.txt
DIR ===== /home/villas/test/dir_2
/home/villas/test/dir_2/text1.txt
/home/villas/test/dir_2/text2.txt
/home/villas/test/dir_2/text3.txt
DIR ===== /home/villas/test/dir_1/dir_1_1
/home/villas/test/dir_1/dir_1_1/text1.txt
DIR ===== /home/villas/test/dir_1/dir_1_2
/home/villas/test/text2.txt

```

A partir da classe VBFileDirectory desenvolvida para unix, pode-se adapta-la para que se possa usar de forma idêntica no Visual C++, importando o componente da seção 17.1.1, com algumas adaptações. Na versão abaixo, a classe VBFileDirectory é adaptada dessas forma, sendo que alguns trechos de programas são compilados de uma ou outra forma controlados pela definição da macro `_MSC_VER`, que somente ocorre no Visual C++.

```

// VBFileDirectory.cpp

#ifdef _MSC_VER // Visual C++ only
#include <windows.h>
#else // unix only
#include <sys/stat.h> // struct stat, S_ISDIR
#include <dirent.h> // DIR
#endif

#include "vblib.h"

class VBFileDirectory {
#ifdef _MSC_VER // Visual C++ only
    WIN32_FIND_DATA m_f;
    HANDLE m_h;
#else // unix only
    DIR *m_dp;
#endif
    VBString m_path, m_dirPath;
    bool m_endOfList;
    bool m_isDir;

    bool p_goodName(VBString fileName) {
        return ((VBString ("..") != fileName) && (VBString (".") != fileName));
    }
}

```

```

void p_isDirectory (VBString file);
VBString p_get();
VBString p_lastFileName(VBString path) {
    unsigned tc = path.tokCount('/');
    path = path.strtok('/',tc);
    tc = path.tokCount('\\');
    path = path.strtok('\\',tc);
    return path;
}
public:
VBFileDirectory () { // default constructor
    m_isDir = false;
}

bool isDir () {
    return m_isDir;
}

VBString getPath () {
    return m_dirPath;
}

VBString getNext() {
    return p_get();
}

VBString getFirst(const char *path) {
    m_path = path;
    m_dirPath = path;
#ifdef _MSC_VER // Visual C++ only
    VBString mask = path + VBString("/");
    mask += "*,*";
    m_h = FindFirstFile(mask, &m_f);
    m_endOfList = (m_h == 0) || (m_h == (void*)0xffffffff);
    if (m_endOfList) return "";
    VBString ret = m_f.cFileName;
    if (!p_goodName(ret)) ret = getNext();
    return ret;
#else // unix only
    m_dp = opendir(path);
    m_endOfList = (m_dp == 0);
    if (m_endOfList) return "";
    return p_get();
#endif
}

bool eolist() {
#ifdef _MSC_VER // Visual C++ only
#else // unix only
    if (m_endOfList) closedir(m_dp);
#endif
    return m_endOfList;
}

};

// set m_isDir true if argument is a directory
void VBFileDirectory::p_isDirectory (VBString file)
{
#ifdef _MSC_VER // Visual C++ only
    DWORD attribs = GetFileAttributes(file);
    m_isDir = (attribs & FILE_ATTRIBUTE_DIRECTORY) != 0;
    // if (attribs == 0xFFFFFFFF) m_isDir=false;

```

```

#else // unix only
    DIR *dp = opendir(file);
    struct stat statbuf;
    lstat(file,&statbuf);
    m_isDir = (S_ISDIR(statbuf.st_mode) != 0);
#endif
    if (m_isDir) {
        m_dirPath = file;
    }
}

VBString VBFileDirectory::p_get()
{
    bool isGoodFileName;
    VBString ret;
#ifdef _MSC_VER // Visual C++ only
    m_endOfList = !FindNextFile(m_h, &m_f);
    ret = m_f.cFileName;
    do {
        isGoodFileName = p_goodName(ret);
        if (!isGoodFileName && !m_endOfList) ret=p_get(); // recursive
    } while (!isGoodFileName);
    ret = p_lastFileName(ret);
#else // unix only
    struct dirent *dirp;
    do {
        dirp = readdir(m_dp);
        m_endOfList = (dirp == 0);
        if (m_endOfList) return "";
        isGoodFileName = p_goodName(dirp->d_name);
    } while (!isGoodFileName);
    ret = dirp->d_name;
#endif
    ret = m_path + VBString("/") + ret;
    p_isDirectory(ret); // set m_isDir
    return ret;
}

```

17.2.2 Entrada cega (útil para entrada de password)

```

// compile with "g++ thisFile.cpp vblib.cpp"
// because this source uses VBString (in vblib.cpp)
// vblib.h also required.
#include <iostream.h> // cout, cin
#include <unistd.h> // read
#include <termios.h> // tcgetattr
#include "vblib.h" // VBString

int main()
{
    // set terminal for no echo
    struct termios myTerm; // a structure to store terminal settings
    int fd = 1; // anything
    tcgetattr(fd,&myTerm); // get terminal settings
    tcflag_t lflagOld = myTerm.c_lflag; // save flag
    myTerm.c_lflag &= !ECHO; // set flag for no echo
    int optional_actions = 1; // anything
    tcsetattr(fd,optional_actions,&myTerm); // set terminal

    VBString password;
    cout << "Enter password:";
    cin >> password;
}

```

```

// restore original terminal
myTerm.c_lflag = lflagOld;
tcsetattr(fd, optional_actions, &myTerm);
cout << endl;
return 0;
}

```

17.3 Elementos de programação em tempo real

17.3.1 Conceitos de programação em tempo real

Até agora, todos os programas são executados com a máxima velocidade que a CPU e o sistema operacional permitem. Em princípio, “quanto mais rápido melhor”. Contudo, há alguns casos em que é importante que se leve em conta o tempo de execução explicitamente para se atingir algum objetivo. Nesses casos não é verdade que “quanto mais rápido melhor”, mas “executar o procedimento no momento certo”. Esses casos são chamados de programação em tempo real.

Um exemplo simples de um programa em tempo real é um programa que mostra um relógio. Um programa desses deve atualizar o mostrador do relógio a cada segundo (ou a cada minuto), no momento certo. A velocidade da CPU precisa ser rápida suficiente para atualizar o mostrador na velocidade solicitada; qualquer velocidade adicional de CPU é irrelevante.

Outro exemplo é um sistema de monitoração de informações baseado em computador ligado a sensores. Num sistema desses, geralmente há um menu de configuração onde se informa quantas vezes por segundo o sensor deve ser amostrado, e seu resultado armazenado (ou mostrado).

Outro exemplo ainda é o caso de um computador que controla uma planta qualquer (como a posição de um canhão ou a posição de um lançador de torpedos). O computador nesse caso é parte integrante de um laço de controle e precisa amostrar os dados do sensor e comandar o atuador em períodos de amostragem pré-definidos. Nesse caso, o computador claramente opera com um programa em tempo real.

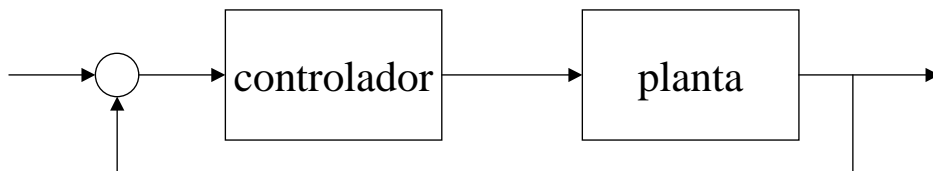


Figura 45: Diagrama do computador como controlador em tempo real

17.3.2 Programa em tempo real com “status loop”

Para um programa operar em tempo real, há uma técnica chamada de “*status loop*”, que em português seria “laço de situação”. Trata-se de um laço de programa que consulta uma base de tempo (que é atualizada por um hardware separado da CPU) para saber se é ou não a hora de executar um procedimento.

O PC é um projeto que contém um “timer tick”, que pode ser consultado. O timer tick é um relógio que interrompe periodicamente a CPU. Isso é muito importante para o sistema DOS, para que se pudesse criar o efeito de multiprocessamento, mas é relativamente pouco importante no Windows, que já possui multiprocessamento internamente.

O DOS inicializa a interrupção do timer tick no boot. Um programa pode capturar

No exemplo abaixo, consulta-se o relógio interno do PC para se saber o instante atual.

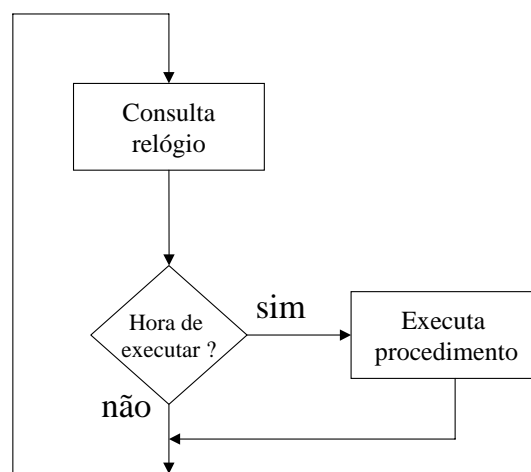


Figura 46: diagrama de um laço para programação em tempo real usando *status loop* (laço de situação)

```

/*=====
NOTE: This is Borland specific source code,
that is, non portable C source code.
=====*/

#include <time.h>
#include <dos.h>
#include <iostream.h>

void t1()
{
    clock_t start, end;
    start = clock();

    // um for para perder tempo
    for (int i=0; i<10000 ; i++)
        for (int k=0; k<10000 ; k++)
            {};

    end = clock();
    cout << "O tempo de espera foi de " << (end - start) / CLK_TCK
        << "segundos" << endl;
}

/*
struct tm {

```



```

    int tm_sec;
    int tm_min;
    int tm_hour;
    int tm_mday;
    int tm_mon;
    int tm_year;
    int tm_wday;
    int tm_yday;
    int tm_isdst;
};

struct time {
    int ti_sec;
    int ti_min;
    int ti_hour;
    int ti_hund;
};
*/

void printLocalTime2()
{
    time_t timer;
    tm *tblock, t;
    timer = time(NULL);    // gets time of day
    tblock = localtime(&timer);    // converts date/time to a structure
    t = *tblock;
    cout << "Hora: " << t.tm_hour << ":" << t.tm_min << ":"
         << t.tm_sec << endl;
}

void printLocalTime()
{
    time t;
    gettimeofday(&t);
    cout << "Hora: " << (int)t.ti_hour << ":" << (int)t.ti_min << ":"
         << (int)t.ti_sec << ":" << (int)t.ti_hund << endl;
}

int main()
{
    float period=1.0;    // in seconds
    float dt;
    clock_t c1, c2;
    c1 = clock();
    while (1) {    // laço infinito
        c2 = clock();    // pega o relógio atual
        dt = (c2-c1)/CLK_TCK;    // calcula o atraso
        if (dt > period) {    // se o atraso for maior que o período
            c1=c2;    // relógio1 é atualizado para a próxima interação
            // a seguir, faz-se alguma coisa em tempos controlados
            cout << "<ctrl-c> to stop" << endl;
            printLocalTime();
        }    // end if
    }    // end while
    return 0;
}    // end main

```

Abaixo há outro programa interessante que lê a hora do sistema. O programa abaixo funciona tanto em Windows quanto em Unix.

```

#include <time.h>
#include <iostream.h>

```

```

#if 0 // commented out begin
struct tm {
    int tm_sec;        // seconds after the minute - [0,59]
    int tm_min;        // minutes after the hour - [0,59]
    int tm_hour;       // hours since midnight - [0,23]
    int tm_mday;       // day of the month - [1,31]
    int tm_mon;        // months since January - [0,11]
    int tm_year;       // years since 1900
    int tm_wday;       // days since Sunday - [0,6]
    int tm_yday;       // days since January 1 - [0,365]
    int tm_isdst;      // daylight savings time flag
};
#endif // commented out end

int main()
{
    char *months[] = {
        "January",
        "February",
        "March",
        "April",
        "May",
        "June",
        "July",
        "August",
        "September",
        "October",
        "November",
        "December"
    };
    char *weekDay[] = {
        "Sunday",
        "Monday",
        "Tuesday",
        "Wednesday",
        "Thursday",
        "Friday",
        "Saturday"
    };

    time_t ltime;
    tm *t;

    time( &ltime ); // get time and date
    t = localtime( &ltime ); // convert to struct tm

    // display
    cout << "Now is:" << endl;
    cout << "Time: " << t->tm_hour << ":" << t->tm_min
        << ":" << t->tm_sec << endl;
    cout << "Year: " << t->tm_year + 1900 << endl;
    cout << "Month: " << months[t->tm_mon] << endl;
    cout << "Day: " << t->tm_mday << endl;
    cout << "Week day: " << weekDay[t->tm_wday] << endl;
    return 0;
}

```

A saída do programa time2.cpp no console é mostrada abaixo.

```

Now is:
Time: 18:25:24
Year: 2000

```

Month: October
Day: 3
Week day: Tuesday

17.3.3 Programa em tempo real com interrupção

O projeto do PC inclui um relógio chamado *timer tick*, que interrompe a CPU regularmente, com frequência de aproximadamente 17 vezes por segundo. O sistema operacional possui uma rotina padrão para tratar essa interrupção. Geralmente, as rotinas de interrupção devem ser muito curtas e como o DOS não é reentrante, é proibido que qualquer função do DOS seja chamada dentro da rotina que trata a interrupção do timer tick.

Para se fazer um programa em tempo real no PC baseado na interrupção do timer tick, é preciso que se escreva uma rotina de interrupção no programa e direcionar a interrupção do timer tick para essa rotina. Para não afetar outras funcionalidades do sistema operacional, a rotina de interrupção deve terminar chamando a rotina original de tratamento do timer tick. Quando o programa termina, a rotina original deve ser restaurada.

O diagrama de um laço de programação em tempo real usando a interrupção do timer tick é mostrada abaixo.

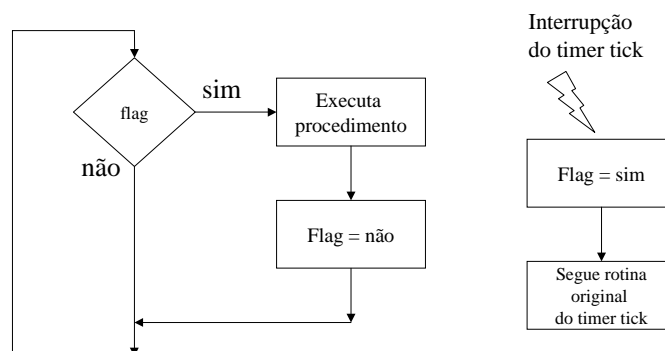


Figura 47: diagrama de um laço para programação em tempo real usando interrupção do *timer tick*

No programa de exemplo abaixo, a rotina de interrupção conta até 17 antes de setar o flag. Com isso, produz-se um efeito como se a rotina do timer tick tivesse uma frequência 17 vezes menor, o que corresponde a aproximadamente 1 segundo. Além disso, o programa redireciona o tratamento de ctrl-c, para que se garanta o restabelecimento da rotina original de timer tick no caso de o programa terminar com o usuário pressionando ctrl-c.

```
/*=====
NOTE: This is an interrupt service routine.
You can NOT compile this program with
Test Stack Overflow turned on and get an
executable file that will operate correctly.
```

```
NOTE2: This is Borland specific source code,
that is, non portable C source code.
```

NOTE3: if you have problems in compiling
this code, try using the command line compiler
bcc.

```

=====*/

#include <stdio.h>
#include <math.h>
#include <time.h>
#include <dos.h>
#include <iostream.h>

#define INTR 0x1C    // The timer tick interrupt

#ifdef __cplusplus
    #define __CPPARGS ...
#else
    #define __CPPARGS
#endif

/*=====
CMAX delta_t(seconds)
17    0.99
18    1.04
=====*/
#define CMAX 17

// global variables
void interrupt ( *oldhandler)(__CPPARGS); // pointer to function
long int count=CMAX;
char flag_timeToRun=0;

#define ABORT 0
int c_break()
{
    char *str="=====";
    cout << endl << str << endl;
    cout << "Control-Break pressed." << endl
         << "Restoring timer tick handler and Program aborting ...";
    cout << endl << str << endl;

    setvect(INTR, oldhandler); // reset the old interrupt handler
    return (ABORT);
}

// don't call DOS inside interrupt handler function
void interrupt interrupt_handler(__CPPARGS)
{
    count++;
    if (count > CMAX) {
        count = 0;
        flag_timeToRun = 1;
    }
    oldhandler(); // call the old routine
}

void printLocalTime()
{
    struct time t;
    gettime(&t);
    cout << "Hora: " << (int)t.ti_hour << ":" << (int)t.ti_min << ":"
         << (int)t.ti_sec << ":" << (int)t.ti_hund << endl;
}

```

```

}

int main()
{
    oldhandler = getvect(INTR); // save the old interrupt vector
    setvect(INTR, interrupt_handler); // install the new interrupt handler
    ctrlbrk(c_break); // register ctrl-c handling routine

    flag_timeToRun = 0; // initialize flag
    for (int i=0; i<15000; i++) {
        for (int k=0; k<30000; k++) { // long loop
            if (flag_timeToRun) {
                flag_timeToRun = 0;
                cout << endl << "CTRL-C to abort    ";
                printLocalTime();
            } // end if
        } // end for k
    } // end for i

    setvect(INTR, oldhandler); // reset the old interrupt handler
    return 0;
} // end main

```

Os programas em tempo real que usam interrupção são um pouco mais problemáticos de se depurar e entender que os programas que usam o *status loop*. Um erro no programa com interrupção pode levar o computador a travar completamente, só se recuperando com um reset geral. Além disso, não se pode rodar debug com um programa que usa interrupção.

Em alguns casos é possível atingir velocidades maiores de execução usando interrupção. Se um problema em questão requer que se use a velocidade de processamento no limite máximo da velocidade, o uso de interrupção pode ser uma boa alternativa. Caso contrário, é mais livre de problemas utilizar a técnica de *status loop*.

17.3.4 Programas tipo “watch dog” (cão de guarda)

Um programa de cão de guarda é um programa que fica “rodando em paralelo” com demais programas com a finalidade de checar a integridade de um sistema. Uma aplicação para esse tipo de programa é gerar um alerta ou alarme no caso em que uma condição seja atingida.

18 Boa programação × má programação

18.1 Introdução

Um compilador C++ é uma ferramenta de propósito absolutamente genérico. Serve para fazer projetos muito sérios, e também para experiências de qualquer natureza. A flexibilidade muito grande que o compilador propicia leva ao programador a desenvolver diversos estilos de programação.

Refletindo sobre os objetivos que uma pessoa tem ao fazer um programa, e revendo os conceitos de programadores experientes, chega-se a um conjunto de regras, chamadas de “regras de boa programação”. Essas regras não são impostas pelo compilador, no sentido que se não forem seguidas, ainda assim o compilador consegue compilar. Mas são regras muito recomendáveis, pois a experiência mostra que a falha em seguir essas regras geralmente causa muitos problemas mais tarde.

Os objetivos de quem programa são mais ou menos os seguintes (não estão em ordem de prioridade):

- Desenvolver programas que solucionem a solicitação de clientes. Faze-lo de forma mais rápida e objetiva possível, evitando-se dentro do possível “re-inventar a roda”.
- Facilitar o provável upgrade, isto é, ser capaz de adaptar um programa, no caso de uma mudança de especificação solicitada pelo cliente. Chama-se isso de “manutenção de programa”.
- Aprender mais sobre a tecnologia de programação. Conhecer as tecnologias em vigor e as tendências.
- Facilitar que outra pessoa, que não necessariamente é conhecida, altere o programa. E essa pessoa pode ser você mesmo após algum tempo.
- Considerar a comercialização do fonte de um programa, e não apenas do executável.

Portanto, desenvolver programas é muito mais que apenas fazer um código que atenda a uma especificação. Caso o programador não se conscientize disso, seu código fica de “baixa qualidade” (o que não quer dizer que não funcione).

Mas os programas que se faz hoje em dia são muito complexos, e é imperioso “jogar em time”. Definitivamente não é mais possível que uma pessoa sozinha faça um programa razoavelmente grande de forma completa, sem nenhum tipo de ajuda, ainda que indireta. Na prática, os programadores trocam muitas

informações uns com os outros. Na era da Internet, essa troca de informações é em grande parte feita por email ou web.

Nesse contexto, é muito importante que o código fonte seja facilmente compreendido pela comunidade de programadores com os quais se pretende trocar informações. Por exemplo: durante o desenvolvimento de um programa, pode acontecer que um trecho de programa simplesmente não funcione. O programador resolve recorrer a uma lista de email para expor o trecho e pedir ajuda. Para fazer isso, a maneira mais fácil é cortar-e-colar o trecho em questão para o email. Mas se o programa está mal programado (no sentido de ser difícil de entender), então será muito mais difícil obter ajuda.

18.2 Itens de boa programação

18.2.1 Identação correta

A identação do programa deve ajudar o programador a entender o nível de laço que se está programando. Quando o programa tiver laços dentro de laços, cada laço deve estar identado em relação ao anterior. O mesmo vale para if. O exemplo abaixo está ilustra o conceito. Repare que a chave que fecha o escopo de for k está exatamente embaixo do for k.

```
void t()
{
    const int row = 5;
    const int col = 15;
    int data[row][col];
    for (int i = 0; i < row; i++)
        for (int k = 0; k < col; k++)
        {
            data[i][k] = i + 10*k + 6;
            if (k == 4)
            {
                data[i][k] = 200; // this case is special
            }
        } // for k
}
```

Os marcadores de escopo ({ e }) devem seguir um dos dois padrões abaixo.

```
if (k==4)
{
    data[i][k] = 200;
}
```

ou

```
if (k==4) {
    data[i][k] = 200;
}
```

18.2.2 Não tratar strings diretamente, mas por classe de string

Uma fonte de bugs muito comum em C/C++ ocorre no tratamento de strings diretamente, ou seja, criar um buffer de char com um tamanho fixo e usar funções de biblioteca padrão que manipulam esse buffer, tais como strcpy ou

strcat. Caso ocorra bug, o comportamento do programa é aleatório, isto é, pode funcionar normalmente, pode funcionar com erro (e.g. copiar a string faltando um pedaço), pode ocorrer mensagem de erro no console, pode travar a máquina, etc.

18.2.2.1 *Motivos pelos quais é má programação tratar strings diretamente*

Fazer um programa que trata strings diretamente é portanto considerado como má programação. Os motivos explícitos de considera-lo assim estão expostos abaixo.

1. O compilador não tem como distinguir entre um ponteiro para buffer, ponteiro alocado e ponteiro não alocado. O programa abaixo contém um bug e o compilador não sinaliza qualquer warning.

```
// programa com bug !
#include <string.h> // strcpy
int main ()
{
    char *buffer=0;
    strcpy(buffer,"abc"); // bug. data is being copied to non allocated buffer
    return 0;
}
```

2. Caso seja usado um buffer de char com tamanho fixo, somente não ocorre bug caso a string nunca ultrapasse o valor desse tamanho de buffer. Portanto o programa sempre tem uma certa possibilidade de contar um bug. Isso é particularmente problemático ao se escrever um componente de programação de propósito genérico. Usando a função strcpy ou strcat, o compilador não verifica se o buffer tem espaço suficiente para a cópia. Caso o espaço não seja suficiente, ocorre bug.

```
// programa com bug !
#include <string.h> // strcpy
int main ()
{
    char buffer[10];
    strcpy(buffer,"abcdefghijklmnpqrst"); // bug. data is larger than buffer
    return 0;
}
```

3. Caso uma função retorne char*, esse retorno não pode ser feito com referência a uma variável local, a menos que seja “static”.

```
// programa com bug !
#include <iostream.h>
#include <string.h>

char *htmlPath(const char *userName)
{
    const char *path = "/home/usr";
    const char *html = "/public_html";
    char buffer[30]; // this line
    strcpy(buffer, path);
    strcat(buffer, userName);
    strcat(buffer, html);
    return buffer; // BUG, returning reference to local variable
}
```



```

}

int main()
{
    char *completePath = htmlPath("fred");
    cout << completePath << endl;
    return 0;
}

```

Esse programa pode ser corrigido acrescentando-se a palavra “static” ao buffer de retorno. Dessa forma, o buffer passa a ter escopo global (e não mais escopo local), apesar de ter visibilidade apenas local. Trocando-se a linha marcada com “this line” pela linha abaixo, obtém-se o efeito desejado.

```
static char buffer[30];    // this line
```

Mas mesmo assim persiste o problema do item anterior, isto é, a função apenas não tem bug se em nenhum caso o conteúdo do buffer ultrapassar o tamanho definido. Pode-se minimizar a possibilidade de bug (não anular essa possibilidade) por fazer o tamanho do buffer ser muito grande. Mas isso faz consumir desnecessariamente recursos do computador.

18.2.2.2 Solução recomendada para tratamento de strings: uso de classe de string

Há mais de uma classe de string que se pode usar. Isso ocorre porque a classe de string não é parte integrante da estrutura da linguagem C++, mas escrita em cima da própria linguagem C++. Portanto, cada programador pode usar/escrever sua própria classe de string de acordo com sua conveniência.

Abaixo, estão citados 3 exemplos de classes de string, com suas características

1. CString - classe que vem junto com a biblioteca MFC e Visual C++. É uma classe muito boa, mas somente funciona em Windows. Não funciona em unix.
2. string - classe padrão de C++, que funciona igualmente em unix e Windows. É bastante boa, mas requer o uso de namespace. Somente compiladores mais modernos dão suporte a namespace (mas isso não é grande problema, porque é muito comum usar sempre as últimas versões dos compiladores). O (pequeno) problema é que nem sempre o programador quer usar namespace, ou sabe fazê-lo.
3. VBString - é uma classe feita pelo Villas-Boas, em código aberto. Para usá-la, é preciso incluir no projeto o arquivo vbllib.cpp, e incluir no fonte vbllib.h. É uma classe de string que funciona igualmente em Windows e unix, e tem várias funções extras (em relação a uma classe de string padrão) que podem ser muito úteis.

Usando classe de string, evita-se o uso de constantes com dimensão de buffer de array of char.

```

// programa sem bug, bem programado
#include "vbllib.h" // VBString
int main ()

```

```

{
    VBString buffer; // não se indica o tamanho do buffer.
    buffer = "abcdefghijklmnpqrst";
    return 0;
}

```

Concatenação de strings pode ser feita diretamente.

```

// programa sem bug, bem programado
#include "vblib.h" // VBString
int main ()
{
    VBString buffer; // não se indica o tamanho do buffer.
    buffer = "abcdefghijklmnpqrst";
    buffer += "123";
    cout << buffer << endl; // abcdefghijklmnpqrst123
    return 0;
}

```

Uma função pode retornar um objeto do tipo string sem problemas

```

// programa sem bug, bem programado
#include <iostream.h>
#include "vblib.h"

VBString htmlPath(VBString userName)
{
    char *path = "/home/usr";
    char *html = "/public_html";
    VBSrtring buffer;
    buffer = path;
    buffer += userName;
    buffer += html;
    return buffer;
}

int main()
{
    VBString completePath = htmlPath("fred");
    cout << completePath << endl;
    // ou
    cout << htmlPath("fred") << endl;
    return 0;
}

```

Uma classe de string possui um buffer interno para armazenar o conteúdo da string. Esse buffer não é acessado diretamente, mas apenas por operadores e outros métodos da interface da classe de string. A classe cuida de alocar o buffer interno na dimensão exata para a medida do que está sendo guardado. Obviamente, a classe também cuida de liberar a memória na destruição dos objetos string, para evitar vazamento de memória. Portanto, um objeto de classe de string pode conter uma string de qualquer tamanho, mesmo que seja patologicamente grande. Essa característica é particularmente útil no tratamento de arquivos de texto, por exemplo. Pode-se ler uma linha de um arquivo de texto diretamente para uma string, despreocupando-se com o tamanho que a linha possa ter.

No programa de exemplo abaixo, um arquivo de texto é lido linha a linha, armazenado numa string e depois enviado ao console. A elegância está no fato

de que não há constante de dimensão de array of char, e o programa funciona mesmo para arquivos que possam ter linhas de dimensão patologicamente grande. Como cada linha é lida e posteriormente enviada ao console, pode-se pensar em fazer algum processamento com essa linha, o que dá maior utilidade ao programa.

É necessário fazer um laço infinito (while true) e parar o laço com “if end-of-file break”, porque em C/C++, o flag de “end-of-file” somente é ativado quando se tenta ler um dado após o último que existe.

```
// programa sem bug, bem programado
#include <fstream.h> // ifstream
#include "vblib.h" // VBString

int main()
{
    ifstream myFile("filename.dat");
    VBString str;
    while (true)
    {
        myFile >> str;
        if (myFile.eof()) break;
        // do some processing, if needed
        cout << str << endl;
    }
    return 0;
}
```

18.2.3 Acrescentar comentários elucidativos

Os comentários devem explicar para que serve um trecho de programa, ou uma função. Os comentários *não* devem explicar a linguagem, a menos em situações específicas (como um tutorial).

```
int main ()
{
    int i;
    // Exemplo de comentário ruim, pois apenas explica a linguagem
    i = i + 1; // soma 1 a variável i
    return 0;
}
```

18.2.4 Evitar uso de constantes relacionadas

Facilitar o upgrade, ao permitir uma nova versão apenas por mudar um único lugar do programa. O programa abaixo é um exemplo de programa mal escrito, pois a constante 5 de `int dataObj[5];` é relacionada com a constante 5 da linha seguinte `for (i = 0 ; i < 5 ; i++)` {. Caso o programa seja alterado e a linha `int dataObj[5];` passe a ser `int dataObj[4];`, o programa passa a estar inadequado, e o seu comportamento é aleatório (pode funcionar perfeitamente, pode não funcionar, pode travar a máquina, etc).

// Exemplo de programa mal escrito por ter constantes relacionadas³²

³² fill dataObj with data = preencher objetoDeDados com dados

```

int main()
{
    int i;
    int dataObj[5];
    // fill dataObj with data
    for (i = 0 ; i < 5 ; i++) {
        dataObj[i] = i + 4;
    }
    return 0;
}

```

A forma correta de escrever esse programa seria como mostrado abaixo. Nesse programa, a dimensão do vetor e a dimensão no qual o vetor é preenchido são relacionadas pela macro `NUM`. Caso seja necessário um upgrade, há apenas uma linha para se modificar.

```

// Exemplo de programa bem escrito por não ter constantes relacionadas
#define NUM 5
int main()
{
    int i;
    int dataObj[NUM];
    // fill dataObj with data
    for (i = 0 ; i < NUM ; i++)
    {
        dataObj[i] = i + 4;
    }
    return 0;
}

```

18.2.5 Modularidade do programa

Um bom programa deve ser “modularizado”, isto é, o programa deve conter muitos “módulos”, que são componentes de programação. O oposto disso é o programa que é todo escrito, de cima a baixo, numa única função, muito longa. Como regra prática, uma função não deve ter em geral muito mais que 30 linhas. Se tiver, provavelmente é mais fácil de ser compreendida se parte dessas 30 linhas transformar-se num “módulo”, isto é, uma função separada.

Outra coisa que se deve evitar é o uso exagerado de loop dentro de loop diretamente. Escrever o código assim dificulta a manutenção do mesmo. Do ponto de vista de legibilidade do código é bastante elegante que o número de níveis de laços dentro de laços (ou if's) seja limitado a cerca de 5 ou 6. Se for necessário mais que isso (digamos 10 níveis), o ideal é criar uma função no meio para “quebrar” a sequência de níveis e facilitar a legibilidade.

Uma outra forma de dizer a mesma coisa é que o programador deve focar a programação mais na biblioteca (conjunto de componentes) e menos na aplicação em si. Os módulos devem ser escritos da forma mais geral possível, de forma que possam ser usados em muitas situações. Assim, esse módulo torna-se um componente de programação. O ideal é que a cada programa feito, o programador aumente seu conhecimento sobre componentes de programação que possui cópia, e que sabe usar.

Caso o programador já tenha componentes de programação (classes, funções, macros, etc.) que estejam sendo usadas por mais de um programa, essas funções devem ser separadas num arquivo separado de fonte (*.cpp) e outro arquivo de header (*.h). A longo prazo, esse arquivo deverá ser a biblioteca pessoal desse programador. Veja o exemplo abaixo. Nesse exemplo é necessário que o projeto contenha os arquivos application.cpp e mylib.cpp. Caso haja uma outra aplicação a ser desenvolvida que use novamente a classe myClass, basta fazer outro arquivo (digamos application2.cpp), usando novamente a mesma classe myClass.

```

////////////////////////////////////
// application.cpp
#include "mylib.h"
int main()
{
    myClass a; // myClass is declared in mylib.h
    a.someFunction(); // the prototype of someFunction() is in mylib.h
    // the code of someFunction() is in mylib.cpp
    return 0;
}

////////////////////////////////////
// mylib.h
class myClass // declaration of myClass
{
public:
    void someFunction(); // prototype of someFunction
    // etc
};

////////////////////////////////////
// mylib.cpp
void myClass::someFunction()
{
    // code of someFunction
}

```

18.2.6 Uso de nomes elucidativos para identificadores

Os identificadores são nomes de funções, de variáveis, de classes, de macros, de métodos, etc. Esses nomes devem significar o que representam. Dessa forma, o programa fica intrinsecamente legível. Algumas regras sobre isso:

- Nomes curtos que não significam nada (e.g. i, k, etc.), podem ser usados para iteradores inteiros simples.
- Atributos de uma classe (variáveis membro) devem começar com m_.
- Nomes com significado devem ser em inglês (veja a sub-seção abaixo).

18.2.7 Programar em inglês

Sem considerações ideológicas, o fato é que inglês é o idioma internacional. Se não o é para todas as áreas, com certeza é para tecnologia da informação.

Portanto, o fato é que os melhores livros, listas de discussão, gurus, páginas web, etc. que falam sobre tecnologia de informação estão em inglês.

Considerando o que já foi dito no sentido da importância de se trocar informações com uma ampla comunidade, de desenvolver e usar bibliotecas, etc. a consequência é que programar pensando em inglês é a forma mais objetiva de se atingir o melhor nível de programação. Nesse idioma será mais fácil obter ajuda, internacional se necessário. Se o seu programa for considerado muito bom, será mais fácil vender o código fonte.

Para quem fala português como nós, pode parecer um aviltamento a recomendação de se programar em inglês. Eu me defendo disso dizendo que a experiência me mostrou que programar em inglês é a forma mais prática de se obter desenvoltura no mercado de tecnologia, que é ultra-competitivo, globalizado e em constante mudança. Eu já tive a oportunidade de examinar programas muito interessantes escritos por programadores que falam alemão, e que falam japonês. Em alguns casos, os programas foram feitos “em inglês”, e em outros casos essa recomendação não foi seguida. Quando a recomendação não era seguida, o programa era útil apenas a nível de execução, mas não a nível de reutilização de código. Se nós programarmos em português, pessoas que não falam português pensarão o mesmo do nosso programa. Na prática, isso significa que o nosso trabalho terá menor valor.

Há que se distinguir muito bem o “profissional de tecnologia de informação” do “usuário” que vai simplesmente operar o programa. O profissional de tecnologia de informação é pessoa em geral muito mais educada que o usuário. Quem tem que falar o idioma nativo é o usuário, não o profissional de tecnologia de informação. O profissional de tecnologia de informação vive num mundo globalizado literalmente, enquanto o usuário geralmente não o faz.

Para dar exemplos concretos: no desenvolvimento do Windows, ou do Linux, ou do Office da Microsoft. Em todos esses casos claramente há um “elemento central” (kernel) do programa, que é feito por profissionais muito qualificados. Com certeza, esses códigos fonte foram escritos em inglês. Mas todos esses programas são usados por usuários no mundo inteiro. Para que os programas possam ser melhor disseminados, são feitas versões deles em vários idiomas. Mas as versões em outros idiomas são meras traduções de tabelas de strings. Não se traduziu o fonte do programa propriamente dito. E estando em inglês, o programa está “bem programado” no sentido que é mais fácil obter ajuda no descobrimento de bugs ou preparação para uma próxima versão.

Programar “pensando em inglês” significa usar nomes de identificadores em inglês, comentários em inglês, nomes de arquivos em inglês, etc. O fonte abaixo está em português, mostrado como exemplo:

```
// em português, má programação !  
// mostra_dados.cpp  
#include <iostream.h>
```

```

#define NUMERO_MAXIMO 5
int main()
{
    int i;
    int meus_dados[NUMERO_MAXIMO];
    cout << "Veja os dados" << endl; // sinaliza para o usuário
    // enche meus_dados de dados
    for (i = 0 ; i < NUMERO_MAXIMO ; i++)
    {
        meus_dados [i] = i + 4;
        cout << meus_dados [i] << endl;
    }
    return 0;
}

```

Esse programa poderia ser melhor escrito assim como abaixo. Repare que apenas os identificadores e comentários foram mudados. A mensagem para o usuário é mantida em português.

```

// in english, good programming !
// show_data.cpp
#include <iostream.h>
#define MAXIMUM_NUMBER 5
int main()
{
    int i;
    int my_data[MAXIMUM_NUMBER];
    cout << "Veja os dados" << endl; // send message to user
    // fill my_data with data
    for (i = 0 ; i < MAXIMUM_NUMBER ; i++)
    {
        my_data [i] = i + 4;
        cout << my_data [i] << endl;
    }
    return 0;
}

```

19 Erros de programação, dicas e truques

19.1 Cuidado com o operador , (vírgula)

Tente completar o programa incompleto abaixo. É preciso que existam 2 funções chamadas f e g. A questão é: quantos argumentos possui a função g ?

```
// add code here
int main()
{
    int a=3, b=4;
    f(a,b);
    g((a,b));
    return 0;
}
```

Resposta: a função g possui 1 argumento apenas, esse argumento é o retorno do operador vírgula, que no caso é o conteúdo da variável b. Seja o programa completo de exemplo abaixo.

```
#include <iostream.h>
void f(int x, int y) {
    cout << "x=" << x << " ; y=" << y << endl;
}
void g(int z) {
    cout << "z=" << z << endl;
}
int main()
{
    int a=3, b=4;
    f(a,b);
    g((a,b));
    return 0;
}
```

A saída do programa é:

```
x=3 ; y=4
z=4
```

19.2 Acessando atributos privados de outro objeto

O encapsulamento pode ser “contornado” se um método de uma classe faz acessar um atributo privado de um objeto recebido como parâmetro. Geralmente isso é má programação, e deve ser evitado. Um exemplo disso é mostrado abaixo.

```
#include <iostream.h>

class myClass {
    int n; // private member
public:
    void fun(myClass & obj) { obj.n=0; } // another object's private member!
    void operator=(int i) { n = i; }
    friend ostream & operator<<(ostream & s, const myClass & obj) {
```



```

        s << obj.n;
        return s;
    }
};

int main()
{
    myClass a,b;
    a = 3;
    b = 4;
    cout << "b=" << b << endl;
    a.fun(b); // a changes b's n
    cout << "b=" << b << endl;
    return 0;
}

```

A saída do programa é:

```

b=4
b=0

```

19.3 Entendendo o NaN

NaN significa *not a number*, ou seja, “não é um número”. Um número expresso em float, double e long double é representado quase sempre com 4, 8 e 10 bytes respectivamente. Quando um número real é representado nesses bytes, o número é convertido para uma representação em ponto flutuante na base 2. Isso é feito com um algoritmo bastante consolidado. Ao se imprimir na tela o número, um outro algoritmo converte o número representado em ponto flutuante na base 2 para uma seqüência de letras, que corresponde ao número na base 10.

Caso, por qualquer motivo, os bytes de um número que deveria ser digamos do tipo float, contém informação binária fora do formato condizente com a representação em ponto flutuante na base 2, então o que se tem armazenado não é realmente um número.

É raro ocorrer um NaN. Em condições normais, nunca ocorre. Contudo, por manipulação de ponteiros, o NaN pode ocorrer (intencionalmente ou não). No exemplo abaixo, um NaN é gerado intencionalmente. Para se testar o NaN usa-se a função `isnan` do unix. No Visual C++, existe a função `_isnan`. Para compatibilizar o código, no exemplo abaixo define-se a função `isnan` a partir da função `_isnan` no caso de o compilador ser Visual C++. Após essa definição, o código pode usar a função `isnan` normalmente. A partir de `isnan`, define-se `testNaN`. Para testar o NaN, inicialmente define-se uma variável tipo float e se lhe atribui uma constante literal. A função `testNaN` diz que o argumento é válido e o envia ao console. Faz-se intencionalmente um type cast inseguro, forçando a um ponteiro para unsigned char a receber o endereço da variável tipo float. Por se preencher byte a byte 4 bytes de dados a partir do endereço apontado pelo ponteiro, como se fosse um array de unsigned char, se está realmente alterando o conteúdo da variável float. Caso o preenchimento de dados seja inconsistente

com a representação em ponto flutuante, o resultado é um NaN, como mostrado abaixo.

```
// nan.cpp
// OK for Visual C++ and g++
// not OK for Borland C++ and Dev-C++

#include <iostream.h>
#include <math.h>

// Visual C++ only
#ifdef _MSC_VER
#include <float.h> // _isnan
int isnan(double d) {
    return _isnan(d);
}
#endif

void testNaN(float z) {
    if (isnan(z)) {
        cout << "argument is not a number" << endl;
    }
    else {
        cout << "argument is " << z << endl;
    }
}

int main()
{
    float a = 10;
    testNaN(a);
    unsigned char *pa = (unsigned char *) &a;
    pa[0] = 0;
    pa[1] = 0;
    pa[2] = 255;
    pa[3] = 255;
    testNaN(a);
    cout << a << endl;
    return 0;
}
```

A saída do programa é:

```
argument is 10
argument is not a number
nan (ou algo parecido, depende do compilador)
```

19.4 Uso de const_cast

Em princípio não se deve converter um parâmetro const para um não const. Mas se isso for necessário, a forma mais elegante de fazê-lo é através de const_cast. O mesmo vale para volatile.

```
#define TYPE int
int main()
{
    const TYPE i = 5;
    TYPE* j;
    j = (TYPE*)&i; // explicit form (not elegant)
    j = const_cast<TYPE*>(&i); // Preferred (elegant)
```

```

        volatile TYPE k = 0;
        TYPE* u = const_cast<TYPE*>(&k);
        return 0;
}

```

19.5 Passagem por valor de objetos com alocação de memória

Quando um objeto possui memória alocada, se esse objeto não tiver construtor de cópia devidamente definido, ocorrem problemas caso seja um parâmetro passado por valor. O problema é que não se aloca memória para o objeto cópia. Portanto, o objeto cópia aponta para a mesma posição alocada do objeto original.

```

#include <iostream.h>

// a class that allocs memory
class myClass {
    double *m_d;
    int m_size;
public:
    myClass(int size=10) {
        m_size = size;
        m_d = new double [1];
    }
    ~myClass() { delete [] m_d; }
};

// parameter by value: bug !
void myFun(myClass c) {
}

// correct version: parameter by reference
void myFun2(myClass & c) {
}

int main()
{
    myClass a;
    myFun(a);
    myFun2(a);
    return 0;
}

```

Quanto um objeto possui memória alocada, é imperioso que se defina explicitamente um construtor de cópia para o objeto. Veja maiores detalhes nas páginas 228 e 248.

```

#include <iostream.h>

// a class that allocs memory
class myClass {
    double *m_d;
    int m_size;
public:
    myClass(int size=10) {
        m_size = size;
        m_d = new double [1];
    }
}

```

```

~myClass() { delete [] m_d; }

// copy constructor
myClass (const myClass & obj) {
    if (obj.m_size > 0) {
        m_d = new double [m_size]; // alloc mem
        for (int i=0 ; i < m_size ; i++)
            m_d[i] = obj.m_d[i]; // copy data contents
    }
}

};

// parameter by value: now ok !
void myFun(myClass c) {
}

// correct version: parameter by reference
void myFun2(myClass & c) {
}

int main()
{
    myClass a;
    myFun(a);
    myFun2(a);
    return 0;
}

```

19.6 Sobrecarga de insersor e extrator quando se usa namespace

Um programa simples que faz sobrecarga de `operator<<` (insersor) ou `operator>>` (extrator) pode não funcionar se passado para uso de namespace. Por exemplo, veja o programa abaixo.

```

#include <iostream.h>

class myClass {
public:
    int i;
    friend ostream & operator<< (ostream & s , const myClass & obj);
};

ostream & operator<< (ostream & s , const myClass & obj) {
    s << "i=" << obj.i;
    return s;
}

void main () {
    myClass c;
    c.i = 4;
    cout << c << endl;
}

```

Esse programa compila e liga, e gera como resultado o que é mostrado abaixo.

```
i=4
```

Mas se o mesmo programa for adaptado para usar namespace, como mostrado abaixo

```
// programa com erro (de ambiguidade)
```

```

#include <iostream>
using namespace std;

class myClass {
public:
    int i;
    friend ostream & operator<< (ostream & s , const myClass & obj);
};

ostream & operator<< (ostream & s , const myClass & obj) {
    s << "i=" << obj.i;
    return s;
}

void main () {
    myClass c;
    c.i = 4;
    cout << c << endl;
}

```

O novo programa, como mostrado acima, compila mas não liga. Durante a compilação, ocorre erro de ambiguidade com o operator<<. Esse erro pode ser corrigido como na versão modificada do programa, mostrada abaixo. Nessa versão, o operator<< é definido dentro da classe myClass.

```

// programa sem erro
#include <iostream>
using namespace std;

class myClass {
public:
    int i;
    friend ostream & operator<< (ostream & s , const myClass & obj) {
        s << "i=" << obj.i;
        return s;
    }
};

void main () {
    myClass c;
    c.i = 4;
    cout << c << endl;
}

```

19.7 Inicializando membro estático de uma classe

Em princípio, não é permitido inicializar um membro estático de uma classe dentro da definição da própria classe.

```

struct myclass {
    static double d = 3.3; // error
    static int array[2] = { 1, 2 }; // error
}

```

Mas a inicialização pode ser feita após a definição da classe, como mostrado abaixo.

```

struct myclass {
    static double d;
    static int array[2];
};
double myclass::d = 3.3; // OK

```

```
int myclass::array[2] = { 1, 2 }; // OK
```

19.8 Alocação de array de objetos não permite parâmetro no construtor

Deseja-se definir uma classe, e passar parâmetros pelo construtor. Até aí, tudo bem. Deseja-se um array estático dessa classe, em que cada elemento recebe um parâmetro pelo construtor. Tudo bem também. Agora: deseja-se alocar um array de objetos dessa classe, sendo que cada objeto alocado deverá receber parâmetros pelo construtor. Isso é um problema, pois **não há sintaxe em C++ para essa operação**.

```
class myClass {
    int m_i;
public:
    myClass() {} // empty default constructor
    myClass(int i) {m_i = i;}
};

int main()
{
    myClass a(2); // one object, parameter by constructor. OK

    // array of objects, parameter by constructor. OK
    myClass b[3] = { myClass(4),myClass(5),myClass(6) };

    // ok, but default constructor is called.
    // There is no syntax for parameters in constructor
    myClass *c = new myClass [3];
    return 0;
}
```

19.9 Ponteiro para função global e ponteiro para função membro (ponteiro para método)

O ponteiro para um método de uma classe *não* é conversível (*cast*) para ponteiro de uma função global com protótipo idêntico. Mas um ponteiro para um método estático pode ser convertido. Contudo, convém lembrar que um método estático é na realidade uma função global com visibilidade local, e portanto não pode acessar implicitamente atributos membros da classe.

Pode-se também definir um ponteiro para função membro atribuir-lhe o valor de uma função membro com protótipo compatível. Mas não se pode chamar a função membro de fora do escopo da classe.

```
// static_member.cpp

#include <iostream.h>

// a global function
int plus_2(int i) {
    return i+2;
}

struct myclass {
    // a method that has the same prototype as the global function
```

```

    int m_plus_2(int i) {
        return i+2;
    }

    // a static method that has the same prototype as the global function
    static int m_plus_3(int i) {
        return i+3;
    }
};

typedef int (*fp_type_global)(int); // type for global function pointer
typedef int (myclass::*fp_type_member)(int); // type for member function pointer

int main()
{
    fp_type_global fp_plus; // a function pointer compatible with the global function
    fp_plus = plus_2; // OK
    int k = fp_plus(4); // call global function through function pointer
    cout << k << endl; // 6

    myclass z;
    // below is error. attempt to attribute pointer of member
    // function to function pointer
    // fp_plus = z.m_plus_2;
    fp_type_member fp_plus_member;
    fp_plus_member = z.m_plus_2;
    k = z.m_plus_2(4);
    // below is error: can not call member function from outside class scope
    // k = fp_plus_member(4);
    cout << k << endl; // 6

    fp_plus = z.m_plus_3; // OK. access static member function
    k = fp_plus(4); // call static member function through function pointer
    cout << k << endl; // 7
    return 0;
}

```

A saída do programa é como mostrado abaixo.

```

6
6
7

```

19.10 Singleton

Em alguns casos, pode ser importante que uma determinada classe seja necessariamente instanciada apenas uma vez, isto é, que haja somente uma cópia do objeto no programa. Algumas pessoas chamam esse tipo de classe de “singleTon”. Para que isso seja possível, é necessário que não exista *default constructor* público. Além disso, é preciso existir uma cópia do objeto na classe, e que um método retorne um ponteiro para esse objeto.

É isso que ocorre no exemplo abaixo. A classe MySingleTon possui *default constructor* como privado (portanto não há *default constructor* público). Além disso, há alguns atributos (no caso `m_i` e `m_d`), e também um método público chamado `GetObject`, que retorna um ponteiro estático para o objeto, que existe de forma estática dentro do método `GetObject`.

No programa principal, não se pode instanciar diretamente essa classe (gera erro de compilação). Mas pode-se pegar um ponteiro para o objeto, e acessar os atributos do objeto. Se existir um outro ponteiro que também pegue o ponteiro para o objeto, estará apontando para o mesmo objeto, como pode ser comprovado.

```
// main

#include <iostream.h>

class MySingleton
{
public:
    static MySingleton & GetObjectRef()
    {
        static MySingleton obj; // the actual object
        return obj;
    }

    // general attributes
    int m_i;
    double m_d;

private:
    // default constructor as private not to allow direct
    // instantiation of this singleton class
    MySingleton() {};
};

int main()
{
    // MySingleton a; // ERROR can not directly instantiate a singleton

    // get a pointer to singleton object
    MySingleton & refObject = MySingleton::GetObjectRef();

    // place some data to singleton attributes
    refObject.m_i = 3;
    refObject.m_d = 4.4;

    // get another pointer to the same object
    MySingleton & refAnotherObject = MySingleton::GetObjectRef();

    // show attributes of object
    cout << "i=" << refAnotherObject.m_i << endl;
    cout << "d=" << refAnotherObject.m_d << endl;
    return 0;
}
```

A saída do programa é como mostrado abaixo.

```
i=3
d=4.4
```

19.11 Slicing em C++

Slicing (fatiamento) é o fenômeno de comportamento indesejado de funções virtuais no caso de se perder informação do atributo de um objeto derivado a

partir da passagem de parâmetro para uma função que tem como argumento o objeto base.

Seja o programa abaixo. O programa compila e liga ? Resposta: sim. Desde que a classe Circle é derivada de Shape, o compilador gera um *default assignment operator* (operator=) e irá copiar os campos da classe base (ou seja, os atributos comuns entre a classe base e a classe derivada). Como o atributo m_iRadius pertence a classe derivada, ele não será copiado pelo *default assignment operator*. Além disso, não há acesso a esse atributo pelo objeto da classe derivada, na função globalFunc.

```
#include <iostream.h>

class Shape {
public:
    Shape() { };
    virtual void draw() {
        cout << "do drawing of Shape (base class)." << endl;
    };
};

class Circle : public Shape {
private:
    int m_iRadius;
public:
    Circle(int iRadius){ m_iRadius = iRadius; }
    virtual void draw() {
        cout << "do drawing of Circle (derived class)."
            << "The radius is " << m_iRadius << endl;
    }
    int GetRadius(){ return m_iRadius; }
};

// a global function, and the base argument is passed by value
void globalFuncByValue(Shape shape) {
    cout << "(by value) do something with Shape" << endl;
    Circle *c = (Circle*)&shape;
    c->draw();
}

// a global function, and the base argument is passed by pointer
void globalFuncByPointer(Shape * shape) {
    cout << "(by pointer) do something with Shape" << endl;
    Circle *c = (Circle*)shape;
    c->draw();
}

int main()
{
    Circle circle(2);
    cout << "=== Circle draw" << endl;
    // call the derived class method directly
    circle.draw();

    cout << "=== by value" << endl;
    // pass a derived object as parameter,
    // when function expects a base object
    // slicing occurs
    globalFuncByValue(circle);
}
```

```

    cout << "=== by pointer" << endl;
    // pass a pointer to derived object as parameter,
    // when function expects a pointer to base object
    // slicing does not occur
    globalFuncByPointer(&circle);
    return 0;
}

```

A saída do programa é como mostrado abaixo.

```

=== Circle draw
do drawing of Circle (derived class). The radius is 2
=== by value
(by value) do something with Shape
do drawing of Shape (base class).
=== by pointer
(by pointer) do something with Shape
do drawing of Circle (derived class). The radius is 2

```

No caso, há comportamento indesejado na função `globalFuncByValue`, pois o `Circle` está sendo traçado como um `Shape`.

O slicing é comum em tratamento de exceções (*exception handling*), quando exceções são capturadas usando-se a classe base.

```

class CException {};
class CMemoryException : public CException {};
class CFileException: public CException{};

try{ /*do something silly*/ }
catch(CException exception) { /*handle exception*/ }

```

Para se evitar o slicing, faça funções globais que chamam ponteiros para classe base (como em `globalFunc2`), e não para objetos da classe base (como em `globalFunc`). Dentro da função, mascare o ponteiro para a classe derivada, como mostrado.

19.12 Uso desnecessário de construtores e destrutores

Não é necessário que se crie construtores e destrutores vazios, como mostrado no exemplo abaixo. Alguns programadores, contudo, gostam de defini-los por clareza de programação.

```

class myClass {
//..
public:
    myClass () {} // empty default constructor
    ~myClass () {} // empty destructor
};

```

A linguagem C++ provê automaticamente 4 métodos, que não precisam ser escritos explicitamente pelo programador (a menos que haja necessidade). São eles:

- default constructor: `myClass();`
- copy constructor: `myClass(const myClass &);`

- assign operator: `myClass operator=(const myClass &);`
- destructor: `~myClass();`

Caso esses métodos não sejam escritos pelo programador, vale o que foi provido automaticamente pela linguagem C++. Mas atenção: caso exista um construtor não padrão numa classe, então, caso o programador não inclua um construtor padrão (mesmo que vazio), para todos os efeitos não há construtor padrão.

```
class myClass {
    int m_i;
public:
    myClass (int i) { m_i = i;} // non default constructor
};

class myClass2 {
    int m_i;
public:
    myClass2 () {}; // empty default constructor
    myClass2 (int i) { m_i = i;} // non default constructor
};

void g() {
    myClass a; // error: no default constructor for this class
    myClass2 b; // OK
}
```

19.13 Dicas de uso de parâmetro implícito

Uma função com argumento (parâmetro) implícito (*default argument*) não pode ter o valor implícito redefinido no mesmo escopo. Por isso ocorre o ERROR 1 no programa abaixo. Mas é possível redefinir o parâmetro implícito desde que seja em outro escopo, como ocorre no OK 1 do programa abaixo.

Pode-se também desabilitar o parâmetro implícito, como ocorre no OK 2 do programa abaixo. Após desabilitar o parâmetro implícito de uma função, a linha marcada com ERROR 2 não compila.

Além disso, pode-se redefinir uma função que não tem parâmetro implícito para que tenha, como ocorre em OK 3 no programa abaixo.

```
#include <iostream.h>

void foo_default_parameter(int i=1) {
    cout << i << endl;
}

// ERROR 1. Can not redefine default parameter in the same scope
void foo_default_parameter(int i=3);

void fun1() {
    void foo_default_parameter(int i=3); // OK 1. redefinition of default parameter
    foo_default_parameter(); // calling function with default parameter 3
}

void fun2() {
    void foo_default_parameter(int); // OK 2. disable default parameter
    foo_default_parameter(10); // ok
}
```

```

        foo_default_parameter(); // ERROR 2. because default parameter is disabled
    }

void foo_no_default_parameter(int i) {
    cout << i << endl;
}

void fun3() {
    void foo_no_default_parameter(int=4); // OK 3. enable default parameter
    foo_no_default_parameter(14); // ok
    foo_no_default_parameter();// calling function with default parameter 4
}

int main()
{
    fun1();
    fun2();
    fun3();
    return 0;
}

```

A saída do programa é como mostrado abaixo.

```

3
10
14
4

```

20 Incompatibilidades entre compiladores C++

20.1 Visual C++ 6.0 SP5

20.1.1 for não isola escopo

A forma correta de se compilar o for é mapea-lo de acordo com o código abaixo (veja página 176).

```
for (<inic> ; <test> ; <set>)
    <statement>;
```

Mapeado para:

```
{
    <inic>;
    while (<test>) {
        <statement>;
        <set>;
    }
}
```

Mas no Visual C++, o mapeamento é feito sem o isolamento de escopo, ou seja, conforma o código abaixo.

```
<inic>;
while (<test>) {
    <statement>;
    <set>;
}
```

Portanto, o código abaixo, que é perfeitamente legal em C++, não compila em Visual C++. Devido ao mapeamento errado, a variável “i” é declarada 2 vezes, o que é ilegal.

```
int main()
{
    for (int i=0 ; i < 10 ; i++)
        cout << "i=" << i << endl;
    for (int i=0 ; i < 10 ; i++) // error, i redefinition
        cout << "i=" << i << endl;
    return 0;
}
```

20.1.2 Comportamento do ifstream quando o arquivo não existe

Quando o visual C++ tenta abrir como leitura um arquivo que não existe, é criado um arquivo com conteúdo zero (se o usuário tiver direito de escrita no diretório). Como o arquivo é criado, o ponteiro para o arquivo retorna válido, isto é, o programa não entra no procedimento para o caso de o arquivo não existir. Isso é um erro do compilador.

```
#include <fstream.h>
```

```
int main()
{
    ifstream myFile("non_existent_file.txt");
    if (!myFile) {
        // handle case when file does not exist
    }
    return 0;
}
```

Pode-se contornar esse erro primeiro testando o arquivo, usando o flag `ios::nocreate`, que força a não criação do arquivo.

```
#include <fstream.h>
int main()
{
    ifstream myFile("non_existent_file.txt",ios::nocreate);
    if (!myFile) {
        // handle case when file does not exist
    }
    return 0;
}
```

20.1.3 `ios::nocreate` não existe quando `fstream` é usado com namespace `std`

No Visual C++, quando um arquivo é criado com `fstream` e namespace, o `ios::nocreate` não compila.

```
#include <fstream>
using namespace std;
int main()
{
    ifstream myFile("non_existent_file.txt",ios::nocreate); // error
    if (!myFile) {
        // handle case when file does not exist
    }
    return 0;
}
```

20.1.4 Compilador proíbe inicialização de variáveis membro estáticas diretamente

```
static const int k = 5; // global, ok

struct A {
    static const int i = 5; // member, error in VC
};
```

20.1.5 “Namespace lookup” não funciona em funções com argumento

O compilador deveria deduzir o namespace (“namespace lookup”) de uma função com argumento no mesmo namespace, mas não o faz em Visual C++.

```
namespace myNamespace {
    struct myStruct {};
    void myFun(myStruct obj);
}

void g() {
    myNamespace::myStruct a;
    myFun(a); // ERROR: 'myFun': undeclared identifier
    myNamespace::myFun(a); // OK
}
```

20.1.6 Encadeamento de “using” não funciona

```
void f();
namespace N {
    using ::f;
}
void g()
{
    using N::f; // C2873: 'f': the symbol cannot be used in a using-declaration
}
```

20.1.7 Erro em instanciamento explícito de funções com template

```
// wrong only for Visual C++ (6.0 SP4)
#include <iostream.h>
template<class T>
void f() {
    cout << sizeof(T) << endl;
}
int main()
{
    f<double>(); // correct output: "8" VC output "1"
    f<char>(); // correct output: "1" VC output "1"
    return 0;
}
```

20.2 Borland C++ Builder (versão 5.0, build 12.34)

20.2.1 Inabilidade de distinguir namespace global de local

```
// error in Borland C++, Visual C++. OK for gnu C++
namespace myNamespace {
    int myFun() {return 1;};
}
using myNamespace::myFun;
using namespace myNamespace;
int main()
{
    int i = myFun(); // Ambiguous overload
    return 0;
}
```

20.2.2 Erro na dedução de funções com template a partir de argumentos const

A dedução de funções com template deveriam omitir a existência de “const” no tipo. O exemplo abaixo mostra que isso não ocorre no compilador Borland C++.

```
template<class T>
void myFun(T x) {
    x = 1; // works
    T y = 1;
    y = 2; // error
}

int main()
{
    const int i = 1;
    myFun(i); // error
    int j = 1;
    myFun(j); // OK
    return 0;
}
```

```
}
```

20.2.3 Erro na conversão de “const char *” para “std::string”

A conversão implícita de argumentos tipo “const char *” para “std::string” falha quando funções em template são instanciadas explicitamente.

```
#include <string>
using namespace std;
template<class T>
void myFun(string & s) {};
int main()
{
    myFun<double>("hello"); // error
    return 0;
}
```

20.3 Gnu C++ (versão 2.91.66)

20.3.1 Valor do elemento em vector<double> de STL

Em visual C++, pode-se escrever o programa abaixo.

```
// in Visual C++
#include <vector>
using namespace std;
int main ()
{
    int val = 1;
    vector<double> vectorObject(5,val); // 5 val, OK
    vector<double> vectorObject2(5); // OK
    return 0;
}
```

em gnu C++ (g++), o compilador não aceita o val.

```
// in gnu C++
#include <vector>
using namespace std;
int main()
{
    int val = 1;
    vector<double> vectorObject(5,val); // error
    vector<double> vectorObject2(5); // OK
    return 0;
}
```

É interessante que o problema somente ocorre quando o tipo é `vector<double>`, mas não ocorre com o tipo `vector<int>`.

21 Bibliografia

21.1 Livros

O livro “Using C++” [1] é bastante bom para o iniciante. Aborda o assunto de forma multiplataforma, mas não dá dicas sobre vários compiladores. A parte de RTTI está boa. A diagramação do livro ajuda a entender os trechos de código fonte. O livro “Turbo C/C++” [2] é um pouco antigo. Fala de C longamente antes de falar de C++. O livro é baseado no compilador já obsoleto Turbo C/C++, que depois transformou-se no Borland C++. A listagem de funções de C é bem completa. Há muitos exemplos de código, e ocasionalmente há um muito útil. É preciso tomar cuidado que nem todo código pode-se usar em outro compilador que o da Borland. Outro livro parecido “Turbo C++ completo total” [47] é em Português, e fala basicamente apenas de C++, e não de C. Eu não gosto desse livro porque traduziu as entranhas dos códigos fonte, o que considero um erro. Há trechos muito bons nesse livro, como o que fala de uso avançado de streams da biblioteca padrão. O livro C/C++ Annotated Archives [5] é cheio de exemplos, com CD incluído. Nesse livro, os exemplos costumam usar namespace std (um estilo de programação). É um livro bom para quem quer ver vários exemplos com fonte.

STL é um assunto particular e avançado dentro de C++. Esse assunto somente deve ser abordado por quem tem boa base na linguagem em si, particularmente em programação genérica (template). O livro “Designing componentes with STL” [3] é um livro denso sobre STL, um pouco difícil de se ler. Mas é muito bom, e explica detalhes que não se encontra facilmente em outros livros. É indicado para quem já sabe sobre STL e quer se aprofundar no assunto. Outro livro bom sobre STL é o [27]. O livro explica bem e dá ótimos exemplos elucidativos, em geral sem usar namespace. Esse livro tem uma figura na sua página 34 que ilustra com maestria os elementos de STL.

O livro original do Kernighan & Richie [4] é uma referência de utilidade histórica. Vale a pena folhear o livro para ver como tudo começou, e como C++ é de nível incrivelmente mais alto que essa primeira versão de C (anterior ao C ANSI). Para estudo propriamente dito não acho um livro muito recomendado. O livro sobre a biblioteca padrão de C [8] é uma referência das funções da biblioteca padrão de C. É uma referência boa, e é um livro barato. Pena que trata apenas de C e não de C++.

O Annotated C++ Reference Manual [6], com co-autoria de Stroustrup - o grande autor da linguagem de C++ - é uma obra de referência, tal como uma gramática de um idioma. Aborda-se detalhes da linguagem em si, sem sequer

usar a biblioteca padrão. O que acontece quando um identificador global é o mesmo de um membro de uma classe base e de uma classe derivada: Qual tem precedência ? Essas e outras perguntas são exploradas nessa obra. É recomendada para quem quer aprofundar-se nos detalhes da gramática do C++.

O livro “Designing OO Software” [11] é uma referência muito boa sobre os conceitos mais abstratos de orientação a objetos. O que é um objeto ? Como passar da abstração para a implementação ? A discussão é feita a nível conceitual, sem descer a exemplos de implementação numa linguagem como C++. Nesse livro, comenta-se do exemplo clássico do projeto de software de uma máquina de ATM (máquina de retirar dinheiro, usada por bancos). O livro “UML Distilled” [9] é bom para abordar UML e orientação a objetos. Mas não desce a detalhes de C++ ou de programação em qualquer linguagem. É um livro fino e tem alguns bom exemplos. É um livro mais para analista que para programador. Para quem quer modelar UML de forma visual, e usar o ROSE (uma ferramenta famosa de engenharia de software da Rational), há o livro em Português [12]. Como o ROSE é visual, o livro é cheio de figuras. Há muito poucas referências a programação não visual. Raramente cita-se o C++. Ainda sobre projetos de sistemas de software, há o livro “Large-Scale C++ Software Design” [34], que discute como projetar um sistema grande tomando C++ como opção tecnológica. É uma leitura para quem quer gerenciar tecnologicamente um projeto de software.

O livro “Internet Agents” [10] é bom para quem quer aprender uns truques sobre a web. Há inúmeros exemplos páginas web, em que se comenta como o sistema é feito. As listagens dos programas cgi no final infelizmente estão em perl, e não em C++. Mas com jeito dá para aproveitar. Para quem quer saber de cookies, um livro em português é o [41]. Esse livro aborda os cookies com diversas linguagens de programação, com ênfase em Perl. Sobre cookies veja também a página na web [64]

Para quem programa para Windows com C++ e MFC, e está num nível intermediário, um livro com dicas e comentários valiosos é o MFC Black Book [13]. Não é um livro para iniciantes em programação para Windows. É um livro para quem está em fase de aprofundamento nas funcionalidades da biblioteca MFC, e nos frameworks que a Microsoft recomenda para programação para Windows. Para ler o livro, assume-se que o leitor já não tem dúvidas básicas sobre a linguagem C++ em si. Mais um livro sobre MFC é o [33]. O livro tem trechos bons, mas um aspecto que eu não gosto nele é o uso desnecessário de array of char, onde na minha opinião deveria estar usando classe de string. Um livro extenso sobre programação para Windows com MFC é o [14]. Esse livro mostra vários aspectos da programação para Windows, incluindo COM, ATL, Internet, etc. É um livro grande e caro para se comprar como investimento, quando se tem disposição de realmente encara-lo. Por varrer suas explicações passo a passo, se está fazendo um bom curso de programação para Windows.

Um outro livro do mesmo gênero é o [15], mais extenso ainda que o [14]. Um livro específico para aprender a programar componentes Windows (COM) é o [29]. O livro fala sobre o conceito de COM, e como fazê-los em Windows com C++ e MFC. Ainda sobre Visual C++ e MFC, há o livro [30], que trata do assunto desde o nível de iniciante, com vários exemplos de programação de caixas de diálogo, controles Windows, etc. Esse é um livro bom para quem sabe C++, tem noção de programação para Windows e quer evoluir vendo muitos exemplos. Mais um livro sobre esse assunto é o [31], que revê historicamente a origem do MFC, e apresenta essa biblioteca. Além disso apresenta funcionalidades de extensão de MFC, inclusive ensinando como fazê-lo. Trata-se de um livro para quem está em nível intermediário de uso de MFC.

Há dois livros bons, embora não muito modernos, sobre algoritmos e estruturas de dados [16] e [17]. Nesses livros, explica-se tipos de dados, listas, pilhas, filas, árvores, recursividade, grafos, etc. Não se usa código, mas pseudo-código em Português. Questões como a previsão teórica do tempo de execução de um algoritmo são abordadas.

Um livro bom sobre XML é o “XML by example” [18], lançado logo após o próprio XML. Os conceitos fundamentais de XML são mostrados. Como XML é processado naturalmente por uma linguagem de programação orientada a objetos, o próprio livro aborda esse assunto. Mas a opção de escolha do livro foi usar java para o processamento. Em [54] pode-se copiar as bibliotecas xerces e xalan, que permitem processamento XML. Há versão java e C++ dessas bibliotecas.

Para quem trabalha com computação numérica e científica, um livro bom é o [19]. Há muitos exemplos e dicas legais. Uma coisa que eu não gosto é que em muitos exemplos há vazamento de memória. Mas isso não invalida o livro. Um outro livro muito bom para programação científica é o [20]. Esse é um livro para quem tem boa maturidade em matemática, e quer exemplos e explicações para traduzir métodos matemáticos para código fonte em C++. Outro livro famoso e muito bom sobre computação científica é o [39], que tem a vantagem de ser oferecido gratuitamente. Esse livro é referenciado num time de autores de primeira qualidade, e dá dicas preciosas para quem se interessa pelo setor. Há uma versão do “Numerical Recipes” para FORTRAN, para C e essa versão, para C++. Uma outra vertente da computação científica é o controle em tempo real e o processamento de sinais. Para quem tem esse interesse, um bom livro é o [32], que revê conceitos básicos de processamento de sinais e processamento em tempo real. O livro tem vários exemplos em C++.

Para programar para Windows, para network, dois bons livros são [21] e [22]. Em ambos fala-se do uso de Windows socket, multi-thread, arquitetura cliente-servidor, RPC (Remote Procedure Call), etc. Um outro livro bom sobre network, focando unix, é o [38]. Nesse livro, há excelentes discussões sobre arquitetura cliente-servidor, com vários exemplos de código em C para unix. E

por falar em unix, um bom livro que ensina todos os truques desse sistema operacional – e como ativa-los em programas em C – é o [25]. Um livro conceitual sobre segurança em redes, criptografia, chave pública e privada, algoritmos de criptografia, e assuntos relacionados é o [23]. O livro é muito bom, mas não chega a escrever códigos de programação em qualquer linguagem. Um livro em português sobre TCP/IP é o [40], que tem explicações boas, mas não desce a detalhes de implementação, e não traz nenhum exemplo com código fonte.

Para quem quer aprender sobre tecnologia de jogos 3D e programação em DirectX (inclui Direct Draw, Direct 3D, Direct Sound, etc), uma alternativa é [24]. Além exibir códigos, o livro de quebra faz uma revisão sobre física e sobre geometria - conhecimentos importantes para quem faz esse tipo de programa.

O livro “Exceptional C++” [26] é um pequeno livro cheio de problemas difíceis, com resposta comentada. Muitos problemas usam programação genérica, com template. É um livro para quem quer se aprofundar em detalhes sofisticados de C++, com ênfase em programação genérica. O livro “C++ Programming Language” [28] é a proposta do autor da linguagem para o seu tutorial. Trata-se de um livro cheio de informações úteis. Mas com todo o respeito ao autor, não creio que seja um bom livro para ser tomado como base de um tutorial. O nível do livro é em geral alto demais para ser considerado como referência para iniciantes.

Para quem quer estudar melhor o relacionamento entre tecnologia de informação e negócios, um livro muito bom é o “Empresa na velocidade do pensamento”, por Bill Gates [42]. Não se trata de uma propaganda de Windows. O livro é MUITO acima desse nível. Trata de como funcionam os negócios, e como devem ser os sistemas de informação para suprir essas necessidades.

[1] Using C++, por Rob McGregor, ISBN 0789716674.

[2] Turbo C/C++ – the complete reference, por Herbert Schildt, ISBN 0078815355.

[3] Designing Components with the C++ STL, por Ulrich Breymann, ISBN 0201178168.

[4] The C programming language, por Kernighan & Richie, ISBN 0131101633.

[5] C/C++ Annotated Archives, por Friedman et al, ISBN 0078825040.

[6] The Annotated C++ Reference Manual, por Ellis e Stroupstrup, ISBN 0201514591.

[7] Turbo C++ completo e total, por Pappas e Murray, Makron Books, CDD-00164291-2136.

[8] A biblioteca standard C, por Plauger, ISBN 8570019025, ISBN do original em inglês 0138380120.

[9] UML Distilled, por Fowler e Scott, ISBN 020165783X.

- [10] Internet Agents, por Cheong, ISBN 1562054635.
- [11] Designing Object-Oriented Software, por Wirfs-Brock et al, ISBN 0136298257.
- [12] Modelagem Visual com Rational Rose 2000 e UML, por Quatrani, ISBN 857393154X.
- [13] MFC Black Book, por Williams, ISBN 1576101851.
- [14] MFC Programming with Visual C++ 6, por White et al, ISBN 0672315572.
- [15] Visual C++ Bible, por Leinecker e Archer, ISBN 0764532286.
- [16] Estruturas de dados e seus algoritmos, por Szwarcfiter e Markenzon, ISBN 8521610149.
- [17] Estruturas de Dados, por Veloso et al, ISBN 8570011679.
- [18] XML by example, por Marchal, ISBN 0789722429.
- [19] C++ and Object Oriented Numeric Computing for Scientists and Engineers, por Yang, ISBN 0387989900.
- [20] Scientific and Engineering C++, por Barton e Nackman, ISBN 0201533936.
- [21] Win32 Network Programming: Windows(R) 95 and Windows NT Network Programming Using MFC, por Davis, ISBN 0201489309.
- [22] Network Programming with Windows Sockets, por Bonner, ISBN 0132301520.
- [23] Network Security: Private Communication in a Public World, por Kaufman, Perlman e Speciner, ISBN 0130614661.
- [24] 3D Game Programming with C++, por Goes e LaMothe, ISBN 1576104001.
- [25] Advanced Programming in the UNIX Environment, por Stevens, ISBN 0201563177.
- [26] Exceptional C++, por Sutter, ISBN 0201615622.
- [27] Stl Tutorial & Reference Guide, por Musser e Saini, ISBN 0201633981.
- [28] The C++ Programming Language, por Stoupstrup, ISBN 0201700735.
- [29] COM Programming by Example, por Swanke, ISBN 1929629036.
- [30] Visual C++ MFC Programming by Example, by John E. Swanke, ISBN 0879305444.
- [31] Extending the MFC Library, por Schmitt, ISBN 0201489465.
- [32] C++ Algorithms for Digital Signal Processing, por Embree e Danieli, ISBN 0131791443.
- [33] Intermediate MFC, por Broquard, ISBN 0138482764.

- [34] Large-Scale C++ Software Design, by Lakos, ISBN 0201633620.
- [35] Programando em C++, por Dewhurst e Stark, ISBN 8570016220.
- [36] Orientação a Objetos em C++, por Montenegro e Pacheco, CDD 001642, Editora Ciência Moderna.
- [37] C++ for Linux. Por Liberty e Horvath, ISBN 0672318954.
- [38] Internetworking with TCP/IP, por Comer e Stevens, ISBN 013260969X.
- [39] Numerical Recipes in C++, por Press, Teukolsky e Vetterling, ISBN 0521750334, parcialmente oferecido gratuitamente em <http://www.nr.com/cpp-blurb.html>.
- [40] Internetworking com TCP/IP, por Bennett, ISBN 857331057X.
- [41] Cookies, por Laurent, ISBN 8572515038.
- [42] Empresa na Velocidade do Pensamento, por Bill Gates, ISBN 8571648700.

21.2 Páginas web

Para quem gosta de navegar em páginas web com artigos sobre C++, um bom ponto de partida é o “C/C++ Users Journal” [55], que tem artigos novos regularmente, com nível muito bom. Outra página muito boa e organizada é a “C++ Resources Network” [52]. Nessa página, há uma referência online sobre C++ muito boa, e também uma interessante lista comentada de compiladores (pagos e gratuitos). Para uma dica diária sobre C++, uma boa opção é [56], que comenta a dica e a classifica de acordo com o nível de dificuldade. Um outro local muito interessante e cheio de informação é o “C++ boost” [61] (*boost* significa “reforço”). Um local muito bom para quem quer ler em Português dicas sobre diversos temas de tecnologia de informação, com ênfase em software gratuito, é o Dicas-L [66], mantido pela Unicamp. Uma qualidade desse site é que há uma lista de email que se pode subscrever, e dessa forma recebe-se diariamente dicas por email. Uma FAQ muito boa sobre C++, mantida por Marshall, pode ser vista em [82].

O site “Visual C Developers Journal” [57] é específico para o Visual C++, com inúmeras informações e artigos para leitura. Para quem quer baixar códigos de exemplo feitos pela comunidade, um site legal é o Code Project [58]. Nesse site o usuário pode introduzir seus próprios artigos e ler artigos escritos por outros membros. Muita gente oferece código de exemplo. Trata-se de um site voltado para Visual C++ e ferramentas Microsoft. Para quem usa ferramentas da Borland, a própria Borland mantém uma página na web para estimular trocas de experiências em [62]. Um outro site com muitos códigos C++ (e outras linguagens também) para copiar é o FreeCode [59], que não é a priori focado em compilador algum. Outro bom local é o codeguru [71], que tem vários artigos e códigos fonte para copiar.

Para quem trabalha com computação numérica no contexto de orientação a objetos, um site excelente é o “Scientific Computing in Object-Oriented Languages” [63]. Ali reúnem-se os principais tópicos de interesse, incluindo divulgação de produtos, seminários, listas de email, bibliotecas, etc. Na mesma linha, esse site [72] tem uma biblioteca multiplataforma para aplicação científica chamada GSL (Gnu Scientific Library), com fonte, manual, etc.

Para copiar arquivos de diversas naturezas, um depósito de software muito bom é o “download” [48]. Outro bom site de depósito de software em geral é o “tucows” [49]. Ainda um site bom para programas em geral, focado no sistema operacional Windows é o “winfiles” [50].

Para quem usa C++ e se interessa por programação web/cgi, eu recomendo o site VBMcgi [45], por motivos que o leitor deve suspeitar. Como está explicado na parte 7 do livro (a partir da página **Error! Bookmark not defined.**), o diferencial da biblioteca gratuita VBMcgi é o fato de isolar o webmaster e o webdesigner. Para quem quer entender de cookies (recurso de programação para web), um bom endereço é o Cookie Central [64].

Para quem se interessa por segurança em redes, há o portal Lockabit [67], mantida pela COPPE-UFRJ. A empresa brasileira Módulo mantém um portal [68] em Português muito bom sobre segurança em redes, com lista de email opcional. Outros dois bons locais para ler em Inglês sobre segurança da Internet são [69] e [70].

Para quem se interessa por negócios em geral pode-se ler em português no site da exame [73] (exceto a primeira página, o conteúdo é restrito a assinantes da revista ou da UOL). Outro site muito bom é o da Isto É Dinheiro [74], que não tem proteção. Os sites da Fortune [75] e o The Economist [76] são muito bons, abordando negócios em geral e economia. O site Fast Company [77] é muito bom, abordando negócios com ênfase em TI. O site Zdnet [78] fala muito bem sobre TI, e ocasionalmente aborda negócios que envolvem TI.

Para quem quer uma ajuda para ler em Português conteúdo da web que originalmente é em Inglês, uma dica ótima é usar o serviço gratuito de tradução do Altavista [79]. Use o serviço assim: copie a URL da página em inglês que quer ler para o campo “Web Page”. Selecione “English to Portuguese”, e clique em “translate” (traduzir). O resultado é a página original traduzida. Pode-se clicar nas links que a navegação prossegue, sempre com tradução. Uma outra forma de se usar o serviço de tradução do Altavista é para quem sabe ler em Inglês, e quer ler conteúdo em outros idiomas. Pode-se por exemplo ler o conteúdo do jornal Le Monde (Francês), ou do Asahi Shinbun (Japonês) traduzido para Inglês. Em todos os casos, a tradução feita por computador ajuda, mas não espere por milagres. Não raro a tradução fica “engraçada” (isto é: estranha). Além disso, como não se pode traduzir imagens, e freqüentemente usa-se imagens contendo

texto – especialmente em menus –, partes da página traduzida não são realmente traduzidas.

Para os interessados em STL (Standard Template Library), um excelente tutorial pela web dessa biblioteca pode ser visto em [83].

[43] Link com referências sobre rpm, <http://rpm.redhat.com/RPM-HOWTO/>

[44] Libtool – ferramenta da Gnu para gerenciamento de bibliotecas – <http://www.gnu.org/software/libtool/>

[45] VBMcgi – biblioteca para programação CGI em C++ – <http://www.vbmcgi.org>.

[46] CPLP – Comunidade dos Países de Língua Portuguesa – <http://www.cplp.org>.

[47] Home page do compilador Dev-C++. <http://www.bloodshed.net/>.

[48] <http://www.download.com/>.

[49] <http://www.tucows.com/>.

[50] <http://www.winfiles.com/>.

[51] MinGW: Minimalist GNU For Windows, <http://www.mingw.org/>.

[52] C++ Resources Network – <http://www.cplusplus.com/>.

[53] <http://www.bruceeckel.com/>.

[54] <http://xml.apache.org/>.

[55] C/C++ Users Journal – <http://www.cuj.com/>.

[56] C++ Tip of the day – <http://www.cplusplus-zone.com/free/tip.asp>.

[57] Visual C Developers Journal – <http://www.vcdj.com/>.

[58] <http://www.codeproject.com/>

[59] <http://www.freecode.com/>

[60] The Universal Library – <http://www.ulib.org/>

[61] C++ boost – <http://www.boost.org/>

[62] <http://community.borland.com/cpp/>

[63] Scientific Computing in Object-Oriented Languages – <http://www.oonumerics.org/>

[64] <http://www.cookiecentral.com/>

[65] <http://www.del.ufrj.br/>

[66] Dicas-L – <http://www.dicas-l.unicamp.br/>

[67] Lockabit – <http://lockabit.coppe.ufrj.br/>

- [68] Módulo – <http://www.modulo.com.br/>
- [69] The center for Internet security – <http://www.cisecurity.org/>
- [70] Common vulnerabilities and exposures – <http://cve.mitre.org/>
- [71] <http://www.codeguru.com/>
- [72] Gnu Scientific Library – <http://www.gnu.org/software/gsl/>
- [73] <http://www.exame.com.br>
- [74] <http://www.istoedinheiro.com.br>
- [75] <http://www.fortune.com>
- [76] <http://www.theeconomist.com>
- [77] <http://www.fastcompany.com>
- [78] <http://www.zdnet.com>
- [79] <http://babelfish.altavista.com/>
- [80] <http://www.apostilando.com>
- [81] Livro ideavirus – <http://www.ideavirus.com/>
- [82] C++ FAQ – <http://www.parashift.com/c++-faq-lite/>
- [83] STL tutorial – <http://www.sgi.com/tech/stl/>