Mário Leite

CURSO BÁSICO DE C PRÁTICO E FÁCIL

EDITORA CIÊNCIA MODERNA

Editor: Paulo André P. Marques
Produção Editorial: Laura Santos Souza
Assistente Editorial: Amanda Lima da Costa
Diagramação: Abreu's System
Capa: Daniel Jara
Copidesque: Kamilla Loivos

FICHA CATALOGRÁFICA

LEITE, Mário.

Curso Básico de C: Prático e Fácil

Rio de Janeiro: Editora Ciência Moderna Ltda., 2013.

1. Programação de Computador – Programas e Dados 2. Ciência da Computação
I — Título

ISBN: 978-85-399-0337-5 CDD 005
004

Editora Ciência Moderna Ltda.
R. Alice Figueiredo, 46 – Riachuelo
Rio de Janeiro, RJ – Brasil CEP: 20.950-150
Tel: (21) 2201-6662/ Fax: (21) 2201-6896
E-MAIL: LCM@LCM.COM.BR
WWW.LCM.COM.BR **10/13**

Dedicatória

Este livro é dedicado aos meus queridos irmãos, *in memorian*:
Antônio Leite, Edith Leite, João de Deus Leite, Lourdes Leite,
Maria da Conceição Leite e Rosária Leite.

Citação

A Educação é tão importante para a segurança nacional, que as aulas deveriam ser ministradas pelo(a) presidente(a) da república .

Agradecimentos

Agradeço primeiramente a Deus, que me deu força e persistência para escrever mais este livro, com muita pesquisa e dedicação, ao longo de dois anos de estudos e pesquisas ininterruptos.

Agradeço a todas às minhas amigas e amigos pela força e incentivo que sempre me deram em todas as minhas obras.

Agradeço aos diretores, colegas professores e pessoal administrativo do CEIT pelo incentivo e apoio incondicional que sempre me deram em todas as minhas publicações.

Agradeço a todos os colaboradores da Editora Ciência Moderna Ltda: aos senhores George Meireles e Paulo André Pitanga, Aline Marques, Camila Cabete, Laura Souza e Amanda Lima pela presteza nos contatos e pela confiança em mais este trabalho.

Sobre o autor

Mário Leite é natural de Tombos (MG), estudou física durante dois anos no Instituto de Física da UFRJ; foi aluno de Iniciação Científica do Centro Brasileiro de Pesquisas Físicas (CBPF) e do CNPq, no Rio de Janeiro. É graduado e pós-graduado em engenharia pela Pontifícia Universidade Católica do Rio de Janeiro (PUC/RJ), onde foi professor auxiliar de ensino e pesquisa no Departamento de Ciências dos Materiais e Metalurgia. É especialista em Análise de Sistemas pelo Centro Universitário de Maringá (CESUMAR) e mestre em Engenharia de Produção pela Universidade Federal de Santa Catarina (UFSC).

Trabalhou durante quatro anos na Indústria e Comércio de Minérios (ICOMI) no estado do Amapá como engenheiro de pesquisas, desenvolvendo aplicações para o setor de produção.

Foi chefe do Setor de Informações Gerenciais da Mineração Caraíba S.A no estado da Bahia durante nove anos, ministrando cursos de técnicas de programação para os engenheiros da empresa e desenvolvendo aplicações para os setores de produção e manutenção. Nessa empresa participou do projeto de mecânica das rochas para implantação do sistema de escavação da mina subterrânea, na adaptação do *software* de elementos finitos para microcomputadores.

É professor de linguagem de programação, análise de sistemas e tecnologia da informação em cursos de processamento de dados, sistemas de Informação e administração de empresas. Também é autor de vários livros na área de computação.

Atualmente é professor da Universidade de Uberaba - UNIUBE - nas áreas de ferramentas computacionais e linguagens de programação. Também é professor convidado do SENAI/CTM de Maringá.

Prefácio

Prefaciar uma obra acadêmica é, antes de tudo, um grande privilégio. Um privilégio que, de algum modo, nos aproxima do autor naquilo que de mais prazeroso existe em todo o processo de elaboração da obra: dar vida pública à criação.

Ao prefaciar uma obra, o fazemos de algo já pronto, já pensado, pesquisado, criado e recriado tantas vezes até que esteja pronto para o fim a que se destina, em nosso caso, para auxiliar o estudante na compreensão das questões da linguagem de programação C.

Nos últimos dois anos acompanhamos a determinação incondicional do professor Mário Leite na pesquisa e elaboração desta obra que ora vem à luz. Sem descuidar da grande responsabilidade de apresentar o resultado das incansáveis pesquisas do nosso mestre, podemos dizer que dividimos, senão o esforço acadêmico do autor, a alegria e o prazer de ver a obra concluída e exposta à crítica da comunidade acadêmica e ao deleite dos aprendizes em buscar nos ensinamentos contidos neste livro o caminho seguro e a necessária luz para o aprendizado da linguagem de programação C.

Vivemos um momento feliz ao testemunhar a conclusão deste longo processo com a publicação dos resultados das pesquisas do professor Mário Leite e, temos a certeza que mais felizes estarão os acadêmicos que terão nesta obra o porto seguro para a resolução dos intrincados problemas tratados nesta obra.

Conhecendo a incansável mente indagadora de pesquisador do professor Mário Leite, já aguardamos o seu anúncio do próximo desafio de sistematização e pesquisa de um novo livro que, como este, se dedicará a auxiliar todos aqueles que despertam, se encantam e fascinam pelo conhecimento.

A todos os leitores, tenham bom proveito e aprendizado sob a "batuta" do nosso mestre Mario Leite.

Lucineia de Caires Bressanim

Diretora do CEIT

Nota do autor

Muito embora já exista no mercado uma gama enorme de bons livros sobre programação em linguagem C, esta obra foi escrita com o objetivo de proporcionar aos programadores iniciantes e intermediários uma leitura mais didática, mais prática e mais agradável sobre o assunto. Talvez, pelo fato de ser uma linguagem muito poderosa e muitas vezes "rebelde", alguns livros que atualmente existem no mercado tratam o C com certa áurea de mistérios; muitas vezes com sofisticações que poderiam ser evitadas, pelo menos para os programadores iniciantes que se assustam com tamanha avalanche de informações. Por isso, muitos desses futuros bons profissionais desistem de aprender C antes mesmo de empreender um estudo mais sério sobre a linguagem. E muito embora isso possa ser discutível por muitas razões, o que se tem estatisticamente é um déficit enorme de programadores (bons) no Brasil. Por isso essa linguagem é fundamental para que o candidato a um profissional de TI (seja desenvolvedor ou programador) se posicione bem no mercado que, mesmo com falta de profissionais, ainda existe concorrência entre aqueles que realmente estão prontos para trabalhar numa empresa de alto nível e exigente.

Este livro busca incentivar o iniciante na linguagem C a olhar essa linguagem como uma aliada na solução de problemas de programação para computadores digitais, sem apresentar soluções sofisticadas, ou aquelas mostradas em alguns fóruns de discussões que parecem "milagrosas", fazendo com que o leitor se sinta inferior perante elas, e às vezes se exclamando: "*...puxa, que fantástico, eu não conseguiria criar essa rotina; esse cara é um gênio!*" Esses

deslumbramentos dos programadores iniciantes são muito comuns e normais, mas o que não se deve perder de vista é que não existe a "melhor solução" para um problema a ser resolvido através da computação eletrônica; o que realmente interessa ao usuário é a solução do problema e não como ele foi resolvido. É claro que um programa deve ser eficiente além de eficaz; e para isto o programador deve estudar e se aprimorar o tempo todo.

Esta obra foi escrita para ajudar programador encarar o C como uma linguagem que pode ser dominada sem recorrer a sofisticações desnecessárias e a rotinas "mirabolantes" sem objetividade no aprendizado. E para isso são apresentados um leque muito amplo de exemplos esclarecedores, esquemas, figuras e mais de cem exercícios propostos e resolvidos.

Os códigos dos exemplos e dos exercícios propostos neste livro poderão ser baixados livremente pelo site da Editora Ciência Moderna no endereço www.lcm.com.br.

Mário Leite

Organização do Livro

Capítulo 1 - Introdução ao C

Neste capitulo inicial o leitor tem uma visão geral da linguagem C; seu histórico, como ela difere de outras linguagens, seu poder de processamento e vantagens sobre as demais, e suas principais características enquanto linguagem de médio nível. Também aqui o leitor iniciante toma conhecimento do que é um código-fonte e como ele é traduzindo para a linguagem de máquina; as diferenças entre compilação e interpretação do código fonte. São apresentados os cinco ambientes principais de desenvolvimento de programas mais utilizados atualmente: Turbo C, C-Free, Visual C++, Code Blocks e Dev-C++. Finalizando o capítulo são propostos oito exercícios para o leitor testar seu aprendizado adquirido.

Capítulo 2 - Iniciando com o C

O segundo capítulo começa explicando em telas, passo a passo, como escrever/editar/salvar/compilar/executar um programa em C. Explica como saber se um programa foi compilado corretamente ou se ocorreu algum erro de sintaxe no código-fonte. Discute o problema de case *sensitive* para os programadores iniciantes e que não estejam familiarizados com esse detalhe que é tão importantes na linguagem C. Mostra os conceitos de comando e linha de instrução; como comentar uma linha ou um bloco de linhas de instruções. Também introduz o conceito de "arquivos de cabeçalhos" (bibliotecas) de funções e sua importância num programa em C. Discute algumas funções que são indicadas na literatura e que não funcionam em todos os compilados como,

por exemplo, clrscr() e gotoxy(), mostrando o porquê isso acontece e ensinando como resolver essas questões. Apresenta também o pré-processador: sua importância e para que serve, além de mostrar a criação de macros e as diretivas mais importantes. Ao final deste capítulo são propostos seis exercícios para treinamento do aprendiz na lnguagem.

Capítulo 3 - Elementos Básicos

Neste capítulo o livro focaliza os elementos básicos da linguagem: tipos de dados, variáveis, constantes, palavras-chave, formatadores de tipos, conversões explicitas e implícitas, sequências de escape, formatadores de saídas, operadores e operações. Também é mostrado como trabalhar com cores e sons nos programas em C. Diversos exemplos são mostrados para cada elemento apresentado, fixando bem o aprendizado. Sobre o assunto "sons" o livro mostra exemplos de como gerar sons usando somente a função Beep(), mas também de uma maneira mais avançada com funções da biblioteca <allegro.h>. Neste item é mostrado como reconfigurar o ambiente do Dev-C++ para instalar a referida biblioteca, mostrando passo a passo como fazer seu *download* e sua instalação para ser incluída nos programas. Exemplos bem interessantes são apresentados com o uso dessa biblioteca para músicas nos formados MIDI e WAVE. Juntamente com o Capítulo 5, este é um dos mais extensos do livro por se tratar dos elementos que são básicos para compreender bem a linguagem C. Dez exercícios são propostos ao final deste capítulo, envolvendo questões objetivas e descritivas sobre os tópicos estudados.

Capítulo 4 - Entradas e Saídas

Este quarto capítulo trata das entradas de dados (pelo teclado) e saídas das informações (no vídeo). São apresentados os conceitos de *stream* para o tratamento de input/output de programas. Aqui são estudadas as principais funções de I/O: scanf(), printf(), getc(), putc(), gets() e fgets(). Também é mostrado como obter precisão em saídas numéricas através de opções de formatação. O capítulo é "recheado" de exemplos e esquemas explicativos para que o programador iniciante entenda perfeitamente como usar as citadas funções. São propostos onze exercícios para testar o aprendizado e solidificar os conhecimentos adquiridos.

Capítulo 5 - Estruturas de Controle

Este capítulo, muito bem planejado, trata dos comandos compostos de controle de fluxo do programa. Apresenta as estruturas de decisão (desvios condicionais if.. e if..else), estrutura de seleção (desvio selecionado - switch), estruturas de repetição (while, do while e for). São apresentados vários exemplos de programas e esquemas explicativos -incluindo diagramas de bloco- para melhor fixar os conceitos. O problema do *falso* e *verdadeiro* também é alvo de uma explicação bem contundente para que o iniciante entenda perfeitamente como isso funciona no C. O operador ternário ? também é estudado como alternativa de compactação do código-fonte em exemplos bem didáticos sem sofisticações desnecessárias. As saídas abruptas de *loops* (*break* e *continue*) são exaustivamente explicadas com exemplos e esquemas, de uma maneira bem didática, mostrando quando e como empregá-las. A função ***exit*()** também é estudada, assim como o comando **goto** que, embora não seja muito usual nos programas, pode ser empregado em situações excepcionais. Este é um dos capítulos mais extensos do livro (juntamente com o Capítulo 3) e fundamental para o programador; por isso ele aborda muitos exemplos, esquemas e figuras para solidificar bem a base da programação em C. Finalizando o capítulo são apresentados doze exercícios propostos para apreciação do leitor.

Capítulo 6 - Trabalhando com Funções

O sexto capítulo apresenta um tema muito importante (e fundamental) para uma boa programação: as funções. O objetivo é mostrar que a modularização de um programa é fundamental para que ele tenha uma melhor legibilidade, além de ser melhor manutenível. Mostra como criar funções nos programa em C, como trabalha a função principal **main()** e sua importância no arquivo--fonte. Explica didaticamente os quatro tipos de funções quando ao retorno e parâmetros. Também discute os tipos de passagem de parâmetros: “por valor” e “por referência” através de esquemas bem explicativos. Mostras os vários tipos de variáveis quanto ao escopo: local, estática e global, e quando e como definir esses tipos de variáveis. A função main() é alvo de um item em particular sobre como executá-la a partir de uma janela DOS recebendo parâmetros na linha de comando. Funções recursivas também são mostradas, e como

criá-las em C a partir de alguma indução matemática, como por exemplo, "Quadrado Perfeito" e "Sequência de Fibonacci". Ao final capítulo são propostos dez exercícios com base também em assuntos dos capítulos anteriores.

Capítulo 7 - Vetores

Os vetores são abordados neste capítulo sob uma ótica bem didática e fácil de entender; com esquemas, exemplos e figuras. O assunto é tratado de maneira a aumentar o nível de compreensão do leitor e baixar o nível de dúvidas que muitas vezes aparecem em materiais sobre programação C. As características de um vetor é explicada de modo bem contundente e sem sofisticações. São mostradas aplicações práticas com vetores: leitura, impressão, ordenação, inversão e pesquisa (seqüencial e binária) baseadas em esquemas explicativos passo a passo em algoritmos/programas fáceis de entender. Ainda neste capítulo o livro mostra como tratar *strings* (cadeias de caracteres) enfatizando essa característica da linguagem com programas-exemplos bem didáticos e comentados, utilizando as principais funções de tratamento de caracteres. São propostos quatorze exercícios ao final do capítulo para o leitor verificar o aprendizado.

Capítulo 8 - Matrizes

Complementando o assunto "arrays" este capítulo trata das matrizes. Aqui elas são apresentadas como uma generalização de vetores de maneira bem fácil de entender, começando bem tranquilo e aumentando o grau de complexidade bem suavemente. Através de figuras e esquemas, as matrizes multidimensionais são mostradas, definidas e inicializadas sem dificuldades para o programador iniciante e intermediário. Declaração, inicialização, leitura e impressão dos elementos de uma matriz são assuntos tratadas de maneira bem "leve", enfatizando as características dos arrays na linguagem C. Operações com matrizes também é um assunto bastante discutido e estudado neste capítulo: adição de matrizes, subtração de matrizes, multiplicação de matrizes por um escalar, produto entre matrizes, determinante de matrizes e passagem de matrizes como parâmetro para uma função. Todos esses assuntos são explicados com programas-exemplos comentados e bem fáceis de entender. Ao final do capítulo são propostos onze exercícios com diversas características (questões objetivas e programas) para o programador treinar o que aprendeu.

Capítulo 9 - Estruturas Complexas

O nono capítulo trata das estruturas em C como complemento de arrays. Os quatro tipos básicos de estruturas são mostrados: **struct, enum, union** e **typedef**, nesta ordem. O objetivo deste capítulo é mostrar como criar uma estrutura heterogênea, em oposição às estruturas homogêneas representadas pelos arrays. É mostrado como criar estruturas (tal como uma tabela em banco de dados) para conter dados de tipos diferentes para uma mesma entidade. Declaração, leitura e atribuição são estudadas através de esquemas explicativos e programas curtos e objetivos. Composição de estruturas, passagem de elementos de uma struct e uma struct inteira para funções, são tópicos estudados de maneira bem tranquila e sem sofisticações. A enumeração (enum) é estudada com exemplos práticos e simples, com bastante objetividade e com exemplos bem esclarecedores. União (union) é mostrada como uma complementação de struct, enfatizando a diferença básica entre elas com exemplos fáceis de entender. Finalmente typedef é apresentado como uma alternativa de melhoria do código-fonte na criação de "novos" tipos de dados, para aumentar a legibilidade dos programas que envolvam estruturas heterogêneas. Este capítulo apresenta ao seu final treze exercícios para o leitor trabalhar os assuntos tratados.

Capítulo 10 - Alocação de Memória

Para os programadores C a alocação de memória é um assunto que não pode faltar em nenhuma obra dessa natureza. Este capítulo aborda com bastante atenção esse assunto com uma abordagem básica, mas muito bem planejada e apresentada através de esquemas e figuras para não deixar quaisquer dúvidas. Primeiramente é mostrado como alocar dinamicamente uma porção de memória utilizando as principais funções para isso: malloc(), calloc(), realloc() e free(), com exemplos práticos e bem didáticos. Em seguida são estudados os ponteiros de uma forma que o programador iniciante vai entender perfeitamente, com esquemas e exemplos bem esclarecedores. Operadores e operações com ponteiros são mostradas em pequenos exemplos com objetivo de solidificar os conhecimentos adquiridos. Ponteiros e vetores, vetores de ponteiros, passagem de ponteiros como parâmetros para funções, ponteiros e *strings*, ponteiros e estruturas e ponteiro de ponteiro; todos esses assuntos são abordados através de

programas-exemplos bem didáticos e comentados. Finalizando os assuntos um exemplo sobre execução de programas externos é apresentado para verificar a alocação de memória através da chamada de um comando DOS. O capítulo é encerrado com a proposta de onze exercícios para o leitor resolver.

Capítulo 11 - Tratamento de Arquivos

Neste capítulo do livro são tratados os arquivos em disco: arquivos-texto e arquivos binários; como acessar, ler e escrever elementos numa estrutura tipo FILE. Primeiramente são estudados os arquivos-texto e em seguida os arquivos binários. A maioria das funções de tratamento de arquivos é aqui apresentada: fopen(), flose(), fgetc(), fputc(), fscanf(), fprintf(), fgets(), fputs(), feof(), fread(), fwrite(), ferror(), remove(), rewind() e fseek(). Por todo o capítulo são mostrados exemplos práticos de acesso a arquivos; no caso de arquivos-textos são mostrados seus conteúdos em figuras que esclarecem objetivamente o que aconteceu depois dos acessos de leitura e/ou gravação. Ao final deste capítulo são propostos cinco exercícios para solidificar os conhecimentos sobre os assuntos tratados.

Apêndice A

Neste apêndice são apresentadas e comentadas as soluções de todos os 111 exercícios propostos ao longo dos onze capítulos.

Apêndice B

Neste apêndice são mostradas algumas funções oferecidas pela linguagem C, de maneira bem resumida; apenas como referência para os programadores.

Apêndice C

Neste apêndice são apresentadas as tabelas ASCII (normal e estendida) com códigos de 0 a 127 e de 128 a 255, respectivamente.

Sumário

Capítulo 1 - Introdução ao C 1

1.1 - Introdução 1

1.2 - Características da linguagem 2

1.3 - Tradução do código-fonte 3

1.4 - Ambientes de desenvolvimento em C 6

1.4.1 - Turbo C 6

1.4.2 - O C-Free 7

1.4.3 - O Visual C++ 8

1.4.4 - Code Blocks 9

1.4.5 - Dev-C++ 12

1.5 - Exercícios Propostos 16

Capítulo 2 - Iniciando com o C 17

2.1 - Criando, compilando e executando um programa 17

2.2 - O problema do *case sensitive* 23

2.3 - Comando e Linha de Instrução 25

2.4 - Comentários 26

2.5 - Bibliotecas (arquivo de cabeçalhos) 27

2.6 - O problema de limpeza da tela 41

2.6.1 - Implementação e uso da função clear() agregada 42

2.6.2 - Criando uma biblioteca com a função clear() livre 43

2.6.3 - Solução mais simples para limpar a tela 45

2.7 - O Pré-processamento 47

2.8 - A importância da endentação 50

2.9 - Exercícios Propostos 54

Capítulo 3 - Elementos Básicos 55

3.1 - Tipos de Dados 55

3.2 - Faixa de valores de um tipo de dados 62

3.3 - Variáveis e Constantes 64

3.3.1 - Variáveis 64

3.3.2 - Constantes 69

3.4 - Palavras-chave 70

3.5 - Formatadores de tipos de dados 71

3.6 - Conversão de Tipos de Dados 73

3.7 - Sequências de escape 75

3.8 - Saída formatada de números reais 77

3.9 - Operadores e Operações 77

3.9.1 - Operadores Aritméticos 77

3.9.2 - Operadores Relacionais 79

3.9.3 - Operadores Lógicos 79

3.10 - Outros operadores 81

3.10.1 - Operadores de incremento/decremento 81

3.10.2 - Operadores de atribuições compactas 82

3.10.3 - O operador de Atribuição 83

3.10.4 - O operador Vírgula 83

3.11 - Precedências dos Operadores 84

3.12 - Tratamento de Cores 86

3.13 - Tratamento de Sons 90

3.14 - A biblioteca <allegro.h> 96

3.14.1 - *Download* e instalação da biblioteca <allegro.h> 97

3.14.2 - Criação de projetos com a biblioteca <allegro.h> 104

3.14.3 - Configurações básicas da biblioteca 106

3.15 - Exercícios Propostos 110

Capítulo 4 - Entradas e Saídas 113

4.1 - Ambientes de entrada/saída de dados 113

4.1.1 - Função scanf() 114
4.1.2 - Função printf() 115
4.1.3 - Função getc() 118
4.1.4 - Função putc() 119
4.1.5 - Função gets() 120
4.2 - Opções de saídas formatadas 127
4.2.1 - Opções de formatação 127
4.2.2 - Largura do campo 128
4.2.3 - Precisão 131
4.2.4 - Tamanho da variável 132
4.3 - Exercícios Propostos 132

Capítulo 5 - Estruturas de Controle 135

5.1 - Conceitos básicos 135
5.2 - Estruturas de Decisão 138
5.2.1 - Estrutura de Decisão Simples 138
5.2.2 - Estrutura de Decisão Composta 140
5.3 - Aninhamento de comandos de decisão 146
5.3.1 - Sobre os Desvios Condicionais 149
5.3.2 - O Operador Ternário ? 150
5.3.3 - Detalhes sobre a expressão condicional 152
5.4 - Estrutura de Seleção 153

5.5 - Estruturas de Repetição 163
5.5.1 - *Loop* lógico com teste no início 163
5.5.2 - *Loop* lógico com teste no fim 170
5.5.3 - Estrutura de repetição numérica 172
5.6 - Saídas de loops (desvios incondicionais) 179
5.6.1 - Comando break 179
5.6.2 - Comando exit() 180
5.6.3 - Comando continue 182
5.6.4 - Comando goto 185
5.7 - Exercícios propostos 187

Capítulo 6 - Trabalhando com Funções 191

6.1 - Modularização de um Programa 191
6.2 - Escopo de Variáveis e Constantes 196
6.2.1 - Escopo Local 196
6.2.2 - Escopo Global 200
6.2.3 - Variáveis Estáticas 202
6.3 - Passagem de parâmetros 204
6.3.1 - Passagem de parâmetros Por Valor 207
6.3.2 - Passagem de parâmetros Por Referência 208
6.4 - Tipos de Funções 210
6.4.1 - Função com retorno e com parâmetros. 211

6.4.2 - Função com parâmetros e sem retorno 212
6.4.3 - Função sem parâmetros e com retorno 212
6.4.4 - Função sem retorno e sem parâmetros 213
6.5 - Sobre a função main() 215
6.6 - Funções Recursivas 219
6.7 - Estilos de programas com funções 226
6.8 - Exercícios Propostos 228

Capítulo 7 - Vetores 231

7.1 - Introdução 231
7.2 - Conhecendo um vetor 231
7.2.1 - Declarações de Vetores 232
7.2.2 - Operações com Vetores 233
7.3 - Aplicações práticas de vetores 239
7.3.1 - Ordenação de vetores 239
7.3 2 - Pesquisas em vetores 245
7.4 - Inversão em vetores 253
7.5 - Tratamento de ***strings*** 256
7.6 - Algumas funções de tratamento de ***strings*** ***258***
7.6.1 - Função strlen() 258
7.6.2 - Função stremp() 258
7.6.3 - Função strepy() 259
7.6.4 - Função streat() 259

7.6.5 - Função toupper() 260
7.6.6 - Função tolower() 260
7.6.7 - Função fgets()....261
7.7 - Exercícios Propostos....265

Capítulo 8 - Matrizes 269

8.1 - Introdução....269
8.2 - Conhecendo uma matriz....269
8.3 - Declarações de Matrizes....270
8.4 - Leitura de Matrizes....272
8.5 - Operações com Matrizes 274
8.5.1 - Adição de matrizes 275
8.5.2 - Subtração de matrizes....277
8.5.3 - Multiplicação....280
8.5.3.1 - Multiplicação de uma matriz por um vetor....280
8.5.3.2 - Multiplicação de uma matriz por um escalar....282
8.5.3.3 - Multiplicação de uma matriz por outra matriz....284
8.6 - Determinante de uma matriz 2x2....286
8.7 - Determinante de uma matriz 3x3....287
8.8 - Detalhes importantes sobre vetores e matrizes....289
8.8.1 - Espaço ocupado por uma matriz na memória....289

8.8.2 - Problema de verificação de limites.....290
8.8.3 - Passando matriz como parâmetros 291
8.8.4 - Recebendo um vetor como retorno de função.....294
8.8.5 - Passando uma matriz como parâmetro de uma função 296
8.9 - Exercícios Propostos.....297

Capítulo 9 - Estruturas Complexas 299

9.1 - Introdução.....299
9.2 - Estruturas struct 299
9.2.1 - Alguns detalhes sobre struct 302
9.2.1.1 - Inicialização de estruturas 302
9.2.1.2 - Leitura de uma estrutura.....302
9.2.1.3 - Exibição de uma estrutura struct 303
9.2.2 - Atribuição entre estruturas struct.....303
9.2.3 - Matrizes de estruturas.....304
9.2.4 - Composição de estruturas.....309
9.2.5 - Passando elementos de uma struct como parâmetros.....311
9.2.6 - Passando uma struct inteira como parâmetro 312
9.3 - Estruturas enum 315
9.4 - Estruturas union.....320
9.5 - Estruturas typedef.....323
9.6 - Exercícios Propostos.....333

Capítulo 10 - Alocação de Memória 339

10.1 - Introdução 339

10.2 - Alocação dinâmica 340

10.2.1 - Função malloc() 341

10.2.2 - Função free() 344

10.2.3 - Função calloc() 344

10.2.4 - Função realloc() 345

10.3 - Ponteiros 346

10.3.1 - Operadores primários de ponteiros 349

10.3.2 - Operações com ponteiros 352

10.3.3 - Ponteiros e Vetores 355

10.3.4 - Vetores de Ponteiros 358

10.3.5 - Passagem de ponteiros para funções 358

10.3.6 - Ponteiros e ***strings*** ***362***

10.3.7 - Ponteiros e estruturas 363

10.3.8 - Ponteiro de Ponteiro 368

10.4 - Executando programas externos 370

10.5 - Alguns ***bugs*** no emprego de ponteiros 373

10.6 - Exercícios propostos 375

Capítulo 11 - Tratamentos de Arquivos 377

11.1 - Introdução 377

11.2 - O tipo de dado para arquivos 377

11.3 - Função fopen() 377

11.4 - Função fclose() 379

11.5 - Função fgetc() 380

11.6 - Função fputc() 382

11.7 - Outras funções de entrada/saída 386

11.7.1 - Função fscanf() 387

11.7.2 - Função fprintf() 388

11.7.3 - Função fgets() 390

11.7.4 - Função fputs() 393

11.7.5 - Função remove() 396

11.7.6 - Função feof() 396

11.8 - Arquivos binários 397

11.8.1 - Funções de manipulação de arquivos binários 397

11.9 - Exercícios propostos 402

Apêndice A - Solução dos Exercícios Propostos 403

Capítulo 1 403

Capítulo 2 405

Capítulo 3 406

Capítulo 4 408

Capítulo 5 414

Capítulo 6 423

Capítulo 7 429

Capítulo 8 439

Capítulo 9 447

Capítulo 10 458

Capítulo 11 465

Apêndice B - Algumas funções do C 471

Apêndice C -Tabelas ASCII 477

Tabela ASCII Normal 477

Tabela ASCII Estendida 482

Bibliografia e Referências Bibliográficas 489

Capítulo 1

Introdução ao C

1.1 - Introdução

A linguagem C é uma linguagem 3GL (de terceira geração), desenvolvida por Dennis Ritchie e Ken Thompson nos laboratórios da AT&T Bell nos EUA por volta do ano de 1972. Historicamente o C é oriundo da linguagem B, que por sua vez foi um aprimoramento da linguagem BCPL, empregada com o sistema operacional UNIX. Como uma alternativa mais rigorosa ao Pascal, o C vem desde então numa curva ascendente de seguidores que a consideram a melhor opção para todo o tipo de solução em termos de programação pura. Esse sucesso se deve em grande parte à sua flexibilidade, à velocidade de seus compiladores, ao conjunto compacto de palavras-chave, à melhor portabilidade, e a uma extensa gama de tipos de dados.

Dentro da classificação quanto ao paradigma das linguagens de programação, o C é uma linguagem imperativa, estruturada e de propósito geral e com recursos quase inesgotáveis. E, embora muitos programadores não gostem dessa classificação, o C é uma linguagem de médio de nível, o que na verdade lhe dá *status* de poder superior às demais ditas de "alto nível" por estar mais perto do *hardware*. Isto quer dizer que ela pode fazer coisas que as outras não conseguem, como por exemplo, a manipulação de *bits*, *bytes* e poder de endereçar a memória mais que as outras linguagens.

O fato mais relevante que alçou a linguagem ao *status* de "queridinha" dos programadores mais experientes, foi o lançamento em 1978 do livro "The C Programming Language", assinado pela dupla Brian Kernighan & Dennis Ritchie, e ficando conhecido no meio dos profissionais de programação como "K&R". Na verdade, a versão descrita da linguagem nessa obra é usualmente referida como "K&R C"; a segunda edição foi inserida num padrão posterior: o ANSI C. O K&R C introduziu algumas características novas na linguagem, tais como: biblioteca padrão de I/O, tipo *struct*, tipos *long int* e tipo inteiro sem sinal (*unsigned int*). Além dessas novidades, alguns operadores foram alterados em função das exigências do compilador.

1.2 - Características da linguagem

Seguindo a marcha inevitável do desenvolvimento da informática, algumas empresas começaram a introduzir mais recursos e novas funcionalidades na linguagem. Destacando-se:

- ❖ Funções sem retorno (***void***);
- ❖ Funções que retornam tipos ***struct*** ou ***union***;
- ❖ Campos de nome struct num espaço separado para cada tipo ***struct***;
- ❖ Atribuição a tipos de dados ***struct***;
- ❖ Qualificadores ***const*** para criar um objeto só de leitura;
- ❖ Biblioteca padrão, que incorpora grande parte da funcionalidade;
- ❖ Enumerações;
- ❖ Cálculos de ponto-flutuante em precisão simples.

Outras características foram incorporadas à linguagem com a padronização na norma ISO 9899:1999 (o chamado C99); esse padrão foi adotado pelo ANSI em março de 1999. Desse modo, as novas características introduzidas foram:

- ❖ Funções em linha;
- ❖ Remoção de restrições sobre a localização da declaração de variáveis;

- ❖ Adição de novos tipos de dados: (long long int, tipo de dados boolean explícito _Bool) e um tipo complex;
- ❖ Vetores de dados de comprimento variável;
- ❖ Suporte para comentários de uma linha iniciados por //;
- ❖ Novas funções de biblioteca; tais como snprintf();
- ❖ Vários arquivos de cabeçalhos, tais como stdint.h.

De um modo geral, a linguagem C possui características que a diferencia de outras linguagens, oferecendo maior poder de implementação.

- ❖ É uma linguagem com um núcleo muito simples, com foco no paradigma de programação procedural e com recursos apoiados em bibliotecas de funções padronizadas;
- ❖ Possui um sistema simples de tipos, evitando operações desnecessárias;
- ❖ Faz uso de um pré-processador para tarefas especiais, como definições de macros e inclusão de múltiplos arquivos-fonte;
- ❖ Usa ponteiros para maior flexibilidade;
- ❖ Permite acesso em baixo-nível através de inclusões de código Assembly no meio do programa C;
- ❖ Parâmetros passados sempre por valor (exceto em vetores simulado através de ponteiros que pode ser passado por referência);
- ❖ Estruturas de variáveis do tipo struct que permitem que dados de tipos diferentes sejam combinados e manipulados como um registro.

1.3 - Tradução do código-fonte

Quando o programador escreve um algoritmo ele está criando uma solução inicial para o problema proposto pelo usuário. O texto formal é o que se chama de **código-fonte**, escrito numa linguagem formalizada e dentro dos padrões rígidos de programação (sua sintaxe). Mas, embora esta formalização represente a solução do problema, esse roteiro ainda não pode ser processado pelo computador; é necessário antes traduzi-lo em **código**

de máquina, o único entendido pelo *hardware*. Essa tradução pode ser de dois tipos:

- ❖ Interpretação;
- ❖ Compilação.

Na Interpretação é feita a leitura/tradução/execução (nesta ordem) de cada linha do programa-fonte, oferecendo ao programador a opção de saber imediatamente se determinada instrução é válida ou não. Isto permite a correção do programa em tempo real.

Na Compilação, após verificação e checagem da sintaxe é gerado um código em linguagem de máquina a partir da tradução integral do programa-fonte, dando como resultado final um outro código **(código executável)** que pode ser executado diretamente pelo sistema operacional.

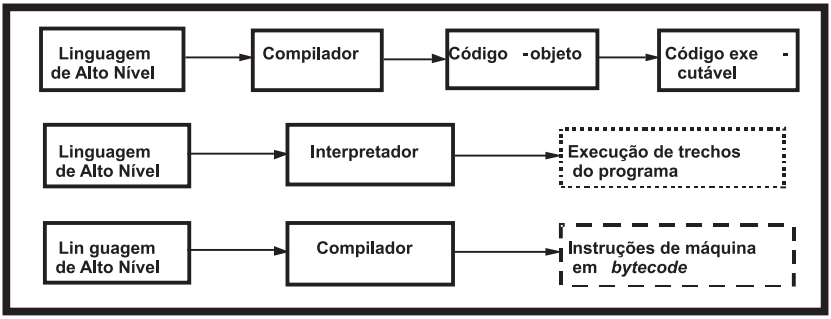
O Quadro 1.1 ilustra como é feita a tradução de um programa escrito em Linguagem de Alto Nível para a Linguagem de Máquina.

Quadro 1.1 - Tradução do programa-fonte em linguagem de máquina

> **Nota:** Após o processo de compilação, normalmente um outro processo se faz necessário: é a *link*edição (ligação). Nessa etapa intermediária (que em algumas ferramentas é transparente ao usuário) são agregadas algumas funções que estão em bibliotecas , produzindo finalmente o arquivo-executável do programa cuja extensão é .exe

O Quadro 1.2 mostra um esquema de tradução do código-fonte escrito em linguagens de alto e/ou médio nível com Compilador e Interpretador. O primeiro esquema mostra que antes de ser obtido o código-executável o processo de compilação cria um arquivo com um código denominado **código-objeto** (transparente ao programador nos modernos ambientes de desenvolvimento) ,

e que ainda contém alguns símbolos legíveis que posteriormente são convertidos totalmente em linguagem de máquina. O segundo esquema mostra como age o interpretador, e o terceiro mostra um sistema de compilação em que são criados *bytecodes* e em seguida estes são interpretados por uma máquina virtual (caso da tradução de um código-fonte escrito na linguagem Java). A Tabela 1.1 mostra as vantagens e desvantagens no uso de cada um desses tipos de tradutores.



Quadro 1.2 - Tipos de tradução do código-fonte

Tradutor	Vantagens	Desvantagens
Compilador	Permite estruturas de programação mais complexas, otimizando o código. Gera um arquivo-executável, permitindo maior autonomia e segurança do código-fonte. Execução mais rápida.	Difícil correção de erros. Não permite correções dinamicamente. Necessita de várias etapas de tradução do códigofonte. Consome muita memória.
Interpretador	Consome pouca memória. Permite estruturas dinâmicas de programação. Tradução em uma única etapa.	Execução lenta. Não gera arquivo executável o que diminui a segurança do código-fonte.

Tabela 1.1 - Comparação entre Compilação e Interpretação

Existem linguagens somente compiladas (C, Pascal, Clipper, Fortran, ...) e somente interpretadas (dBase, Perls, PHP, Python, Foth, Lua, Ruby,...) e algumas com essas duas características juntas num mesmo ambiente, como o Basic e Visual Basic (até a versão 6). Essas últimas produzem melhores resultados, pois na fase de desenvolvimento e depuração o programador pode usar

o Interpretador e na fase final de implantação do sistema usar o Compilador para gerar o código executável, depurado e isento de erros. Entretanto, para muitos programadores essa característica *dual* não é importante, eles argumentam que isso o torna o profissional meio preguiçoso (!).

1.4 - Ambientes de desenvolvimento em C

Após ter desenvolvido o algoritmo da solução do problema, o programador deve converter esse roteiro num programa: o **código-fonte**. A escrita de um programa em C é feita num editor de texto puro, não formatado; por exemplo, usando o Bloco de Notas do Windows ou qualquer outro editor que forneça arquivo em texto ASCII puro. Desse modo, não é uma boa ideia escrever um programa usando processadores de texto como o MS-Word e WordPerfect (dentre os processadores pagos), o que também vale para os gratuitos como BrOffice.Org Writer, AbiWord, Aiword, Atlantis Nova ou o pacote Ashampoo Office 2010, etc. Nenhum desses deve ser usado para digitar o código-fonte de um programa qualquer, pois podem inserir caracteres estranhos no texto (até invisíveis) e gerar erros de compilação. Portanto, para criar o código-fonte deve ser empregado sempre um editor que grave somente em texto puro.

1.4.1 - Turbo C

A Borland International foi a pioneira na criação de ambientes integrados de desenvolvimento de programas com seus famosos **turbos** (quase uma marca registrada no meio dos programadores). Esse ambiente traz um editor, um *linkeditor* (transparente ao usuário), *debugger*, etc. Tudo em um só ambiente de maneira a facilitar o desenvolvimento, a edição e a depuração de programas.

O Turbo C foi introduzido no mercado em 13/05/1987 como o primeiro ambiente edição-compilação-execução para C. A partir daí muitos programadores passaram a utilizá-lo para desenvolver seus programas, pois era (ainda é) uma ferramenta de baixo custo com um arquivo relativamente pequeno. A Figura 1.1 mostra o seu IDE (***I**ntegrated **D**evelopment **E**nvironment* – Ambiente de Desenvolvimento Integrado).

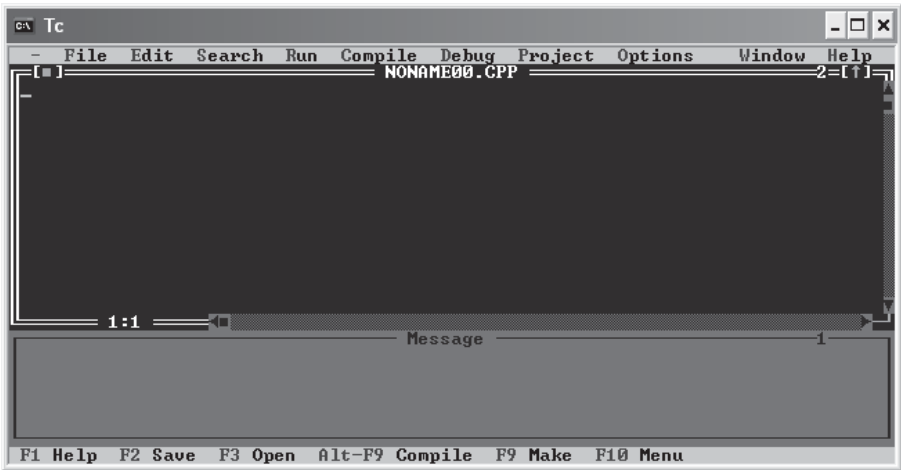


Figura 1.1 - Tela inicial do ambiente de desenvolvimento do Turbo C

Observe na Figura 1.1 que o nome padrão do arquivo-fonte a ser criado tem a extensão CPP (de **C P**us **P**lus - C++) e não apenas .C. Isto é devido ao fato de em 1990 a Borland ter adotado a extensão CPP, pois essa variante do C (C com características de orientação ao objeto) tinha sido introduzida com muito sucesso no mercado. Mas, embora essa extensão sempre apareça atualmente em outros ambientes de desenvolvimento em C, na hora de salvar o arquivo-fonte o programador tem a opção de fazê-lo como .C normal. Isto é, os programadores bastante experientes poderão criar programas em C ou em C++ no mesmo ambiente de desenvolvimento.

1.4.2 - O C-Free

Embora uma cópia *trial* desse compilador possa ser baixada pela Internet, é preciso obter licença para usá-lo na criação profissional de sistemas em C a partir do seguinte *link*: http://www.programarts.com/cfree_en/index.htm (acesso 04/07/12 – 17:25). E, também seguindo a tendência mundial, é um ambiente integrado C/C++ com um compilador muito rápido e IDE muito atraente, oferecendo opções de cores para as palavras-chaves da linguagem e em outras expressões no código-fonte.

A Figura 1.2 mostra o IDE do C-Free. Observe que, como aconteceu com o Turbo C, o padrão sugerido para os programas é C++; entretanto, também podem ser criados programas em C; é o que os programadores fazem.

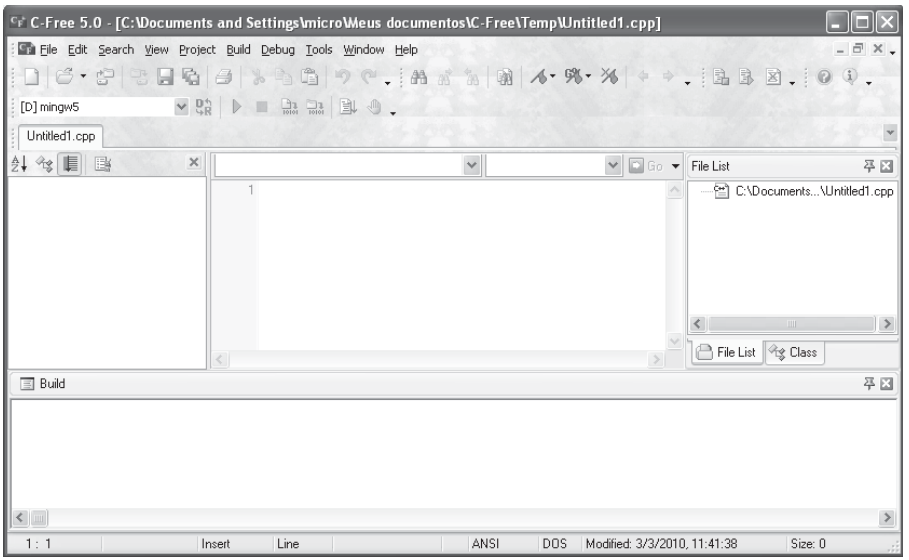


Figura 1.2 - O ambiente de desenvolvimento do C-Free

1.4.3 - O Visual C++

A Microsoft Corporation© também oferece ambientes de desenvolvimento para a linguagem C. O MSC era o seu carro-chefe nessa área; entretanto, com a popularização do C++ a empresa, assim como fez a Borland, optou por oferecer o C dentro do ambiente do pacote Visual C++. Como já foi dito, além do Bloco de Notas do Windows ou outro editor de texto ASCII puro, existem vários editores de texto puro fornecidos com ferramentas que também incorporam o compilador, *linkeditor* e outras funcionalidades, criando um ambiente integrado de desenvolvimento. Assim, é fortemente aconselhável que o programador adote um desses ambientes para a criação, edição e desenvolvimento de seus programas. Desse modo, é possível desenvolver programas em C dentro desses ambientes de maneira visual, dentro da filosofia atual de produtos denominados *studios*. A Figura 1.3 mostra a tela do editor do Microsoft Visual Visual C++.

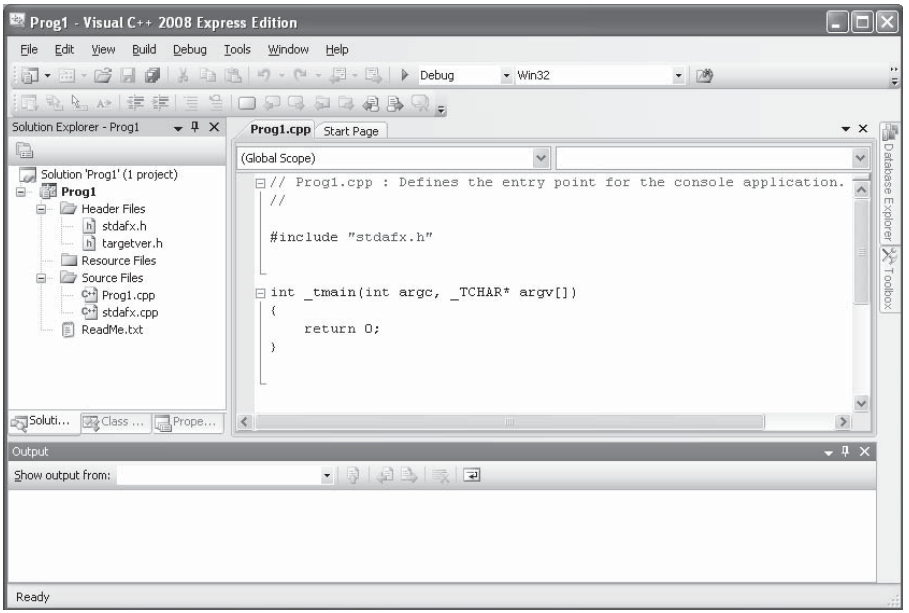


Figura 1.3 - O ambiente de desenvolvimento integrado do MS-Visual Studio C++

1.4.4 - Code Blocks

Code::Blocks (ou C::B) é um ambiente de desenvolvimento integrado de código aberto, multiplataforma e que detecta o compilador instalado (padrão C/C++), e que está sendo muito usado por programadores mais experientes. E como possui um *framework* de *plugins*, o programador mais exigente poderá melhorar a funcionalidade dos seus programas. Esse ambiente roda sob Windows e Linux, o que o torna muito apreciado pelos profissionais de programação. Nas plataformas FreeBSD e Mac OS X alguns problemas de interface podem ocorrer, mas isto é facilmente contornável; e nas versões mais recentes esses "probleminhas" já estão sendo superados. Uma das vantagens de Code::Blocks é que sua IDE vem com uma ferramenta gráfica para a montagem das telas dos programas. A tela da Figura 1.4a mostra a janela inicial dessa ferramenta quando é carregada pela primeira vez.

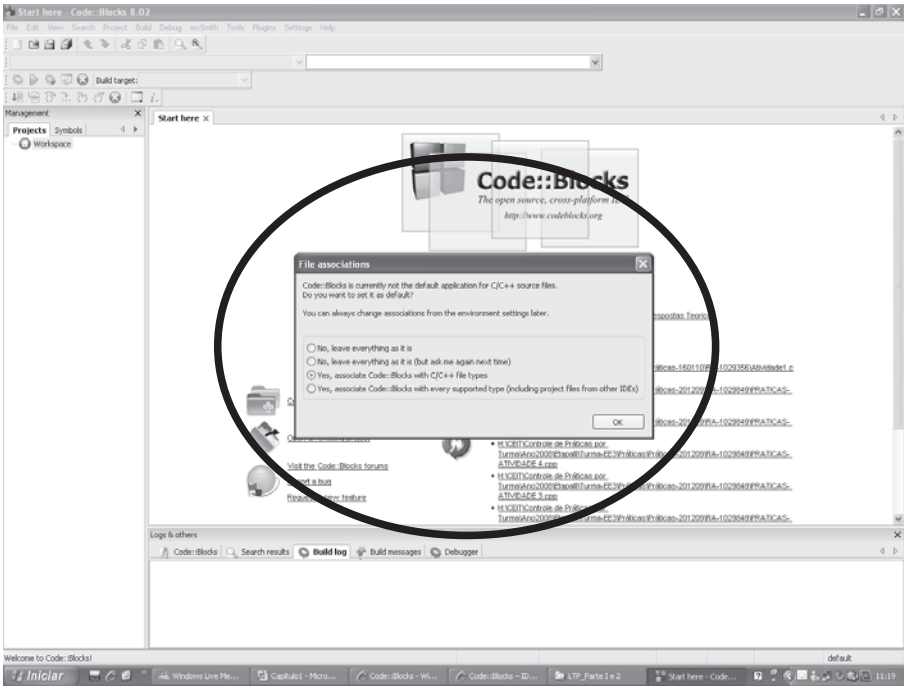


Figura 1.4a - Janela inicial na carga do Code::Blocks

Observe na Figura 1.4a o destaque dado à janelinha, agora ampliada na tela da Figura 1.4b. Nessa janelinha o usuário-programador poderá configurar o ambiente para ser associado ao compilador C/C++ ou deixá-lo livre para outros eventos; esta é uma das vantagens oferecidas por esse IDE.

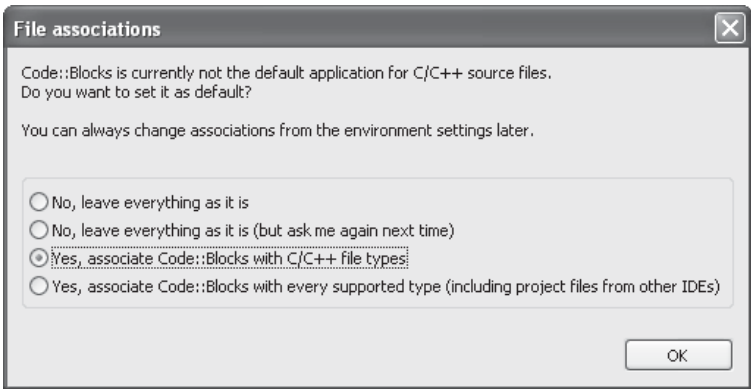


Figura 1.4b - Janelinha de configuração da IDE do Code::Blocks