

Introdução

Aqui introduzimos o funcionamento básico dos computadores e ensinamos como podemos utilizá-los e programá-los para realizar tarefas de nosso interesse.

Algoritmos

Um algoritmo é uma seqüência de instruções que permite realizar uma determinada tarefa. O conceito de algoritmo também é conhecido por outros nomes, entre eles podemos citar: receitas, procedimentos ou técnicas.

Recordando

Em geral,

Algoritmo é uma seqüência finita de instruções necessárias para se realizar uma determinada tarefa.

As instruções que compõem o algoritmo são executadas uma de cada vez, seguindo-se a ordem especificada na seqüência dada. Normalmente, inicia-se a execução pela primeira instrução, exceto quando outra instrução é explicitamente indicada como sendo a primeira a executar. As instruções devem ser sempre primitivas, no sentido de que cada uma delas é individualmente executável pelo computador, não dependendo de interferência de outros agentes. No nosso caso, ao escrevermos nossos algoritmos na linguagem C, esta condição estará automaticamente satisfeita, pois que a linguagem C só conta com instruções que já são suficientemente primitivas, como veremos.

Durante a execução do algoritmo, uma de suas instruções instruí-lo a violar a ordem sequencial, fazendo o ponto de execução saltar para outra instrução que não seja a próxima na seqüência. Por exemplo, pode voltar para a primeira instrução, ou ignorar a próxima instrução passando a executar a segunda seguinte. Normalmente, um salto está associado com uma condição como, por exemplo, uma comparação entre dois números. Nesses casos, realizar o salto dependerá se a condição é verdadeira ou falsa nesse instante. Também é possível que uma instrução simplesmente indique um salto para outra instrução, independente de condições associadas.

Em algum momento, o algoritmo deve encontrar uma instrução sinalizando o fim da execução. Ou seja, o tempo total necessário para terminar a execução do algoritmo deve sempre ser finito e, preferencialmente, dentro de um prazo razoável.

Para ser de utilidade, o algoritmo precisa produzir algum resultado, ou dado de saída. Caso contrário, o algoritmo não nos forneceria nenhuma informação nova e não existiria interesse pelo estudo do mesmo. Por exemplo, ele poderia calcular uma fórmula complexa ou localizar alguma informação em um banco de dados.

Usualmente, algoritmos também aceitam dados como entrada, que são usados para guiar sua execução. Senão, ele sempre produzirá os mesmos dados de saída, ou resposta.

Exemplo de algoritmo

Um dos algoritmos mais conhecidos é o “Algoritmo de Euclides”. Ele é utilizado para determinar o máximo divisor comum (MDC) entre dois números inteiros positivos.

O MDC é determinado realizando divisões entre os dois números. Caso a divisão resulte em resto não nulo, a divisão é novamente realizada, substituindo os números anteriores pelo divisor e pelo resto. Descrevendo essa idéia como uma seqüência de passos elementares, poderíamos ficar com o seguinte algoritmo:

ALGORITMO:

1. Leia dois números.
2. Divida o primeiro pelo segundo e guarde o resto.
3. Se o resto for 0 (zero),
então escreva o segundo número e PARE.
4. Substitua o primeiro número pelo segundo.
5. Substitua o segundo número pelo resto da divisão.
6. Retorne ao passo 2.

Vamos seguir o algoritmo e calcular o MDC entre 24 e 18, seguindo o algoritmo acima. À esquerda está indicado qual passo do algoritmo está executando:

Passo 1: Lê os dados de entrada: 21 e 15.
Passo 2: $21 \div 15 = 1$, resto 6.
Passo 3: Resto é 0? Não, continua com passo 4.
Passo 4: Substitui 21 por 15.
Passo 5: Substitui 15 por 6.
Passo 6: Retornar para passo 2.
Passo 2: $15 \div 6 = 2$, resto 3.
Passo 3: Resto é 0? Não, continua com passo 4.
Passo 4: Substitui 15 por 6.
Passo 5: Substitui 6 por 3.
Passo 6: Retornar para passo 2.
Passo 2: $6 \div 3 = 2$, resto 0.
Passo 3: Resto é 0? Sim. Escreve 3 e PARE.

Ou seja, segundo o valor gerado pelo algoritmo, o MDC entre 21 e 15 é 3, o que está correto.

Note que:

- O algoritmo iniciou sua execução pela primeira instrução (passo 1) e realizou um número finito de instruções até encontrar a instrução PARE (num total de 13 passos executados).
- A entrada do algoritmo foram os números 21 e 15. A saída foi o número 3.
- O passo 3 é uma instrução que contém a condição de teste "Resto é 0?". Se essa condição não é satisfeita, o algoritmo segue com a próxima instrução. Caso contrário, executa o resto da instrução do passo 3 e pára.

- O passo 6 é uma instrução de desvio incondicional, ou seja, ao executar esse passo, o algoritmo simplesmente salta de volta para o passo 2.

Dificuldades na descrição de um algoritmo

A descrição das operações envolvidas em um algoritmo gera várias dificuldades quando utilizamos a língua portuguesa. Devemos considerar:

- As instruções não podem permitir **ambigüidades** na interpretação de sua operação. Alguém poderia questionar o que significa exatamente “Divida o primeiro número pelo segundo e guarde o resto”. O que fazer com o quociente? Trata-se de uma divisão com casas decimais? Como proceder quando os números são negativos?
- Cada instrução deve realizar **uma única operação** bem definida. No caso do algoritmo para calcular o MDC, um dos passos diz: “escreva o segundo número e PARE”. Claramente, a instrução PARE não está relacionada com a instrução ESCREVA.
- As operações devem ser tecnicamente possíveis de serem executadas dentro de um prazo de tempo razoável. Podemos utilizar a operação de divisão, pois existem circuitos elétricos capazes de determinar rapidamente o resultado da divisão de um número por um outro. No entanto, um algoritmo não poderia pedir a fatoração de um número em um único passo, pois não se conhece uma forma eficaz de realizar esta operação.

Linguagem de Programação

Essas dificuldades não podem persistir quando se descreve um algoritmo para execução em um computador, pois este não tem a capacidade de desambiguar as instruções que não estão claras, nem de executar instruções que não estejam pré-definidas em seus circuitos internos. Para aliviar esses problemas quando se descreve algoritmos que vão executar em um computador, buscou-se outras formas para descrever algoritmos.

A primeira alternativa seria uso de recursos gráficos, como, por exemplo, **diagramas**. No entanto, esta abordagem é muito trabalhosa e difícil de automatizar no computador. Assim, predominou uma forma mais prática, as **linguagens de programação**.

Essas linguagens possuem um vocabulário restrito (normalmente palavras do idioma inglês) e várias regras indicando várias restrições que devem ser obedecidas ao se escrever algoritmos. Seguindo-se essas regras à risca, idealmente, toda sequência de instruções descreverá um algoritmo muito bem definido. De forma geral, uma linguagem de programação impede que a pessoa escreva uma instrução de forma ambígua.

Linguagens de programação permitem a descrição precisa de um algoritmo.

Existem programas de computador capazes de ler um algoritmo escrito em uma linguagem de programação e informar se as regras da linguagem foram violadas, impedindo assim qualquer possibilidade inconsistência ou ambigüidade.

Neste curso, estudaremos a linguagem de programação denominada “Linguagem C”, ou simplesmente “C”.

Exemplo

Nesse ponto, limitar-nos-emos a ilustrar algumas características do algoritmo para calcular o MCD quando escrito na linguagem C. Estudaremos os detalhes durante o curso.

Notamos que as operações do algoritmo agora são expressas através de operações aritméticas indicadas por símbolos como ('=', '==', '%'). No entanto, os símbolos que representam as operações não necessariamente são os mesmos com os quais estamos acostumados.

Existem palavras com significado especial na linguagem de programação (chamadas palavras chave) tais como: 'int', 'scanf', 'do', 'break' e muitas outras que aprenderemos durante o curso.

O exemplo também ilustra um caso das regras de uso da linguagem C, também chamadas de regras de sintaxe: todas as instruções devem ser terminadas por ponto-e-vírgula.

Além disso, nem todas as linhas da descrição correspondem a ações a serem executadas pelo algoritmo. Por exemplo, a primeira linha apenas declara os nomes 'a', 'b' e 'r'. A primeira instrução a executar está localizada, na realidade, na segunda linha.

A segunda e penúltima linhas são instruções de entrada e saída de dados, respectivamente. A última linha é a instrução de parada.

Algoritmo 2 – MDC na linguagem C

```
int a, b, r;
scanf("%d %d", &a, &b);
do {
    r = a % b;
    if (r == 0) break;
    a = b;
    b = r;
};
printf("O MDC é %d", b);
return;
```

O computador

Os componentes que formam um computador podem ser classificados em duas categorias: **armazenamento** e **processamento**. O primeiro grupo envolve componentes que guardam dados a serem utilizados pelo computador. Alguns exemplos são: memórias, discos magnéticos, CDs e fitas magnéticas. O segundo grupo é formado por componentes capazes de utilizar os dados armazenados no computador e realizar alguns tipos de operações sobre os mesmos, como localizar informações de um cliente, gerar relatórios, calcular estatísticas ou atualizar os dados armazenados.

Novos dados ou comandos para controlar o computador são obtidos através de dispositivos de “entrada” (teclado, *mouse*, conexões de rede, *scanner*, câmera digital, etc). Outros dispositivos, chamados dispositivos de “saída”, permitem ao computador divulgar dados obtidos pelo processamento ou contidos no armazenamento. Exemplos são: monitor, impressora, caixas de som.

O computador **armazena** dados e realiza **operações** sobre os mesmos. Dados são obtidos por dispositivos de **entrada**. Resultados são informados em dispositivos de **saída**.

Instruções de máquina

Para realizar operações sobre os dados contidos nos dispositivos de armazenamento, o processamento do computador segue um algoritmo, ou seja, uma seqüência finita de instruções bem definidas. Em contraste com as instruções escritas em uma linguagem de programação como C, porém, as instruções diretamente executadas pelo computador estão escritas em outro dialeto, chamado de instruções, ou código, de máquina. Essas instruções são muito mais primitivas que aquelas que podemos utilizar ao descrever algoritmos em linguagens de programação como C. Infelizmente, o *hardware* do computador não consegue executar diretamente as instruções escritas em C. Por outro lado, felizmente, existem meios automáticos de se converter uma seqüência de instruções escritas na linguagem C para instruções de máquina. Dessa forma, os programadores não precisam usar código de máquina diretamente para descrever seus algoritmos, o que seria um esforço muito mais severo e muito mais suscetível a erros. Basta usar instruções da linguagem C e, uma vez pronto o algoritmo, acionar os mecanismos automáticos de conversão.

O código gerado é um algoritmo que o computador consegue executar de forma monótona e repetitiva, porém em altíssima velocidade, simulando as instruções que havíamos escrito na linguagem C.

O computador é uma máquina **rápida e eficiente** para simular algoritmos.

Exemplo

A título de ilustração, apresentamos, nos *slides* que acompanham este texto, um algoritmo para encontrar o MDC de dois números escrito na linguagem de máquina específica de um computador de processador Intel Pentium.

Note como as instruções são pouco intuitivas, pois não conseguem expressar diretamente nossa capacidade de abstrair o problema. Ademais, cada instrução do algoritmo original exige várias instruções de máquina para que se produza o mesmo efeito, ou seja, o código de máquina é uma descrição extremamente detalhista.

Para complicar ainda mais a descrição do algoritmo, o código de máquina exige instruções que só dizem respeito ao circuito eletrônico específico de um tipo de processador. Neste caso, além do programador preocupar-se em escrever corretamente o algoritmo, ele é forçado a planejar seu algoritmo respeitando as características internas de um determinado processador. Muito provavelmente, o algoritmo em código de máquina escrito para um determinado processador, não vai executar corretamente em outro processador.

Níveis de Abstração

Durante esta introdução, apresentamos diversas opções para se descrever um algoritmo. São elas: português, linguagem de programação, e código de máquina.

Os dois extremos são português e código de máquina. A vantagem de se utilizar português está na facilidade de escrever o algoritmo com palavras de nosso dia a dia. No entanto, este texto será potencialmente ambíguo e não há ferramentas para traduzi-lo diretamente para a linguagem do computador.

Já a linguagem de máquina está justamente num formato que pode ser interpretado diretamente pelo computador. Além disso, as instruções são precisas e bem definidas, tal como desejado para se descrever algoritmos. Infelizmente, é muito complicado escrever um algoritmo em código de máquina, pois o resultado é extremamente complexo e detalhista, exigindo um conhecimento profundo sobre as características do processador. Com isso, acaba-se usando tempo e energia no esforço com preocupações irrelevantes à programação de algoritmos.

Buscamos uma opção que permita escrever os algoritmos sem ambigüidade e com operações bem definidas. Esta opção nos é dada pelas linguagens de programação, que oferecem um meio termo interessante entre não ambigüidade e poder de expressão. Existem centenas, senão milhares, de linguagens de programação em uso. Nesse texto, vamos nos concentrar numa das mais populares, a linguagem C.

Antes, entretanto, vamos examinar em um pouco mais de detalhes a tarefa de preparar um programa, ou algoritmo, em uma linguagens de programação.

Construção do Programa

A preparação de um programa, ou algoritmo, para execução em um computador envolve vários passos. O objetivo final é obter um arquivo com código de máquina que possa ser diretamente interpretado pelo computador como um programa ou aplicação:

1. O programador deve, primeiro, elaborar um algoritmo considerando todas as questões estudadas anteriormente, principalmente eficácia e correção. O programador pode representá-lo da forma que preferir.
2. Em seguida, o programador escreve o mesmo algoritmo utilizando uma linguagem de programação. A esta descrição damos o nome de código fonte. Neste curso, aprenderemos a linguagem C para este propósito.
3. Depois, é necessário utilizar um compilador para, automaticamente, traduzir o programa escrito em C para um conjunto de instruções que formam o código de máquina. Para isso, temos vários compiladores disponíveis. Este curso tem como foco principal o compilador C da Microsoft, chamado Visual C.net.
4. Depois de compilar o algoritmo, devemos testá-lo, executando o código de máquina no computador.

Eventualmente, ao escrevermos o algoritmo na linguagem de programação, cometemos erros que serão acusados pelo compilador. Neste caso, voltamos ao passo 2 para corrigir o código fonte do algoritmo.

Durante a execução do algoritmo, eventualmente, encontraremos algumas inconsistências no seu comportamento. A origem do erro exigirá um retorno ao passo 1, para reprojeter o algoritmo, ou também ao passo 2, para reescrever seu código fonte.

Primeiro Programa

Este capítulo apresenta os principais conceitos encontrados na estrutura de um programa de computador escrito na linguagem C.

O programa utilizado como exemplo neste capítulo serve como ponto de partida para os demais exemplos e exercícios ao longo do curso.

O programa "Bom Dia"

Este é um programa bem simples, cujo propósito limita-se a imprimir a frase “O primeiro programa lhe deseja um bom dia!”. Este programa é tão simples que nem sequer possui uma interface gráfica convencional, com a qual estamos acostumados. Ele abre uma janela própria e, portanto, executa dentro de uma janela do *prompt* do MS-DOS.

```
// ProgramaBasico.c: Nosso primeiro programa em C
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char* argv[]) {
    printf("O primeiro programa lhe deseja um bom dia!");
    return 0;
}
```

Observação: O VisualStudio reconhece o código C enquanto você o digita no editor. Ele atribui cores automaticamente ao código C e insere tabulações de acordo com convenções estabelecidas para a linguagem C. As cores e as tabulações podem variar entre os editores. Não há problema quando o seu código apresenta cores ou alinhamento diferente dos exemplos apresentados neste curso, desde que o código seja exatamente o mesmo.

Estrutura do Código Fonte

Já aprendemos que utilizamos a linguagem de programação (neste caso, a linguagem C) para descrever como realizar as operações definidas por um algoritmo. No entanto, para se obter um programa funcional, além da descrição do algoritmo em si, precisamos indicar outras informações. Normalmente, encontramos os seguintes itens em um programa típico. Cada um será descrito detalhadamente mais adiante:

- Comentários
- Diretivas de compilador
- Declaração de variáveis e definição de procedimentos e funções
- Instruções
- Pontuação

Comentários

Comentário é um texto que será ignorado pelo compilador, mas contém informação útil para o programador. Neste exemplo, utilizamos uma linha de comentários para descrever do que trata nosso programa.

O primeiro tipo de comentário é delimitado pelos pares de símbolos “/*” e “*/”, e pode ocupar uma ou mais linhas no código fonte. Já o segundo tipo de comentário, começa com o par de símbolos “//” e estende-se até o final da linha.

Diretivas de compilador

No início do código fonte, antes de qualquer instrução do algoritmo, usualmente é necessário listar algumas diretivas de compilador. Seu uso mais recorrente é indicar outros arquivos que devem ser consultados antes de se iniciar o processo de compilação. Estes arquivos contêm declarações das instruções complexas (i.e. funções) mais comuns, como por exemplo, leitura e escrita de dados em arquivos. Por exemplo, a instrução `printf` não é uma instrução primitiva. Na verdade, trata-se de uma função relativamente complexa. Para que possa ser usada nesse ponto do programa, o compilador tem que conhecer detalhes dessa função. Esses detalhes estão inseridos no arquivo `stdio.h`. É por isso que devemos usar uma diretiva do compilador para inserir esse arquivo antes do uso dessa função no código que estamos escrevendo.

Outro uso das diretivas é alterar o comportamento do compilador para um determinado trecho do código fonte, como veremos ao longo do curso.

Posteriormente serão dados mais detalhes sobre outros usos das diretivas de compilador. Por ora, para entender os exemplos e compilar os exercícios, os programas devem apresentar sempre as seguintes diretivas:

```
#include <stdio.h>
#include <stdlib.h>
```

Caso seja necessário realizar operações matemáticas avançadas, deve-se incluir a seguinte diretiva:

```
#include <math.h>
```

Variáveis, Procedimentos e Funções

Um procedimento ou uma função é um conjunto de instruções que realizam alguma tarefa. Desconsiderando os comentários e as diretivas de compilador, o código fonte constitui-se numa sequência de definições de procedimentos e funções. Além dos procedimentos e funções podemos também listar declarações de variáveis globais. Por ora, vamos ignorar a lista de variáveis, que será tratada oportunamente mais adiante.

Em um programa escrito em C, existe uma função especial, chamada `main`. Esta função contém as instruções por onde se inicia a execução do programa. O capítulo “Procedimentos e Funções”, a ser estudado mais para o final do curso, aborda este assunto em detalhes.

Por enquanto, é importante saber que todo programa em C apresenta uma função exatamente de acordo com o esqueleto abaixo:

```
int main(int argc, char* argv[]) {
    ...
    Algoritmo
    ...
    return 0;
}
```

A declaração entre parênteses após a palavra `main` deve ser `"int argc, char* argv[]"`. Embora pudesse ser omitida em programas mais simples, vamos já incluir essa

declaração, visto que a inclusão não causa danos ao funcionamento do programa. O algoritmo propriamente dito está entre as chaves externas { e }. Por ora, a última linha do algoritmo deve ser `return 0;` para que o programa funcione adequadamente.

Instruções e Pontuação

Cada procedimento ou função é composto por uma lista de variáveis locais e por um conjunto de instruções. Nesse programa simples, não temos uma lista de variáveis locais. Na verdade, a lista de variáveis não faz parte do algoritmo propriamente dito; lidaremos com esse aspecto de variáveis locais e memória logo em seguida.

Já o conjunto de instruções compõem o algoritmo propriamente dito. As instruções são executadas sequencialmente, e cada instrução é separada uma da outra por sinais de pontuação. A pontuação mais comum é o ponto e vírgula (;), mas existem outros símbolos que servem de pontuação: ! % ^ & * () - + = { } | ~ [] \ ' : " < > ? , . / #.

Práticas de programação

Um bom programador, além de preocupar-se com um código fonte, descrevendo um algoritmo de forma correta e eficiente, também considera a aparência visual do código fonte, um fator importante para que outras pessoas possam compreender o código sem muito esforço.

As práticas e estilos aqui recomendados facilitam o entendimento do código fonte.

Linhas em branco. São ignoradas pelo compilador. Utilize para separar blocos de instruções.

Espaços e tabulações. São ignorados pelo compilador. Podem ser úteis para formatar o código fonte. Recomenda-se escrever todas as instruções começando na mesma coluna. Utilize espaços em branco ou tabulações para recuar as instruções um pouco para a direita, diferenciando-as do cabeçalho do procedimento ou da função. Note que a maioria dos editores para a linguagem C fazem isso automaticamente para você.

Uma instrução por linha. Através do emprego correto de pontuação, o compilador é capaz de reconhecer várias instruções em uma mesma linha. Mesmo assim, recomenda-se escrever apenas uma instrução por linha, terminando-a sempre com ponto e vírgula.

No decorrer deste curso, serão apresentadas outras boas práticas de escrita de código fonte.

O Primeiro Programa em Visual Studio.net

Já examinamos o primeiro programa escrito em C que servirá de ponto de partida para todos os demais exemplos e exercícios do curso. Agora, aprenderemos como utilizar o compilador Visual C++ pertencente à ferramenta Visual Studio.net da Microsoft.

Sobre o Visual Studio

O Visual Studio.net é um conjunto de ferramentas para construir aplicações para diversas plataformas, começando de aplicações *desktop* até aplicações corporativas e aplicações para a *internet*. Ele inclui três dos compiladores mais populares: Visual Basic, Visual C/C++ e Visual C#. Além dos compiladores, o Visual Studio conta com uma ampla biblioteca de ferramentas de programação.

O Visual Studio também possui facilidades como editor de texto integrado com o compilador, ajuda interativa e ferramentas de depuração e identificação de erros no código.

Gerenciamento do Código Fonte com Visual Studio

Conforme aumenta o tamanho de um programa e o número de pessoas envolvidas no desenvolvimento do mesmo, seu código fonte estará distribuído por vários arquivos. Para gerenciar eficientemente estes arquivos requeridos pelo projeto, o Visual Studio trabalha com dois conceitos: **soluções** e **projetos**.

Projetos são um conjunto de arquivos de código fonte que, quando compilados juntos, formam um programa. No nosso exemplo (primeiro programa) do capítulo anterior, o programa é formado por um único arquivo de código fonte. O projeto também armazena configurações referentes à compilação dos arquivos e sobre as ferramentas necessárias (como por exemplo, bibliotecas e interface gráfica).

Em um desenvolvimento real, trabalha-se simultaneamente na codificação de vários programas inter-relacionados (ou, usando os termos do Visual Studio: vários projetos inter-relacionados) que, juntos, formam a aplicação. O conjunto de projetos é denominado **solução**. A Figura 1 ilustra estes conceitos:

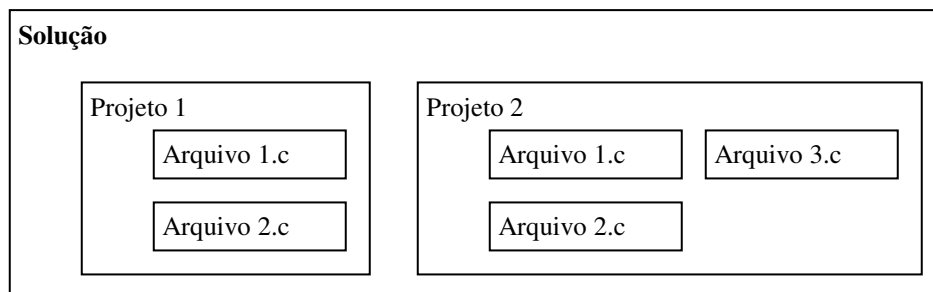


Figura 1 – Hierarquia do código fonte no Visual Studio

Os exemplos apresentados neste curso e os exercícios propostos no decorrer dos próximos capítulos são casos bem simples de projetos e soluções. Por motivos de simplicidade, utilizamos apenas um arquivo de código fonte para formar um projeto que, por sua vez, será o único projeto da solução, conforme pode ser visto na Figura 2.

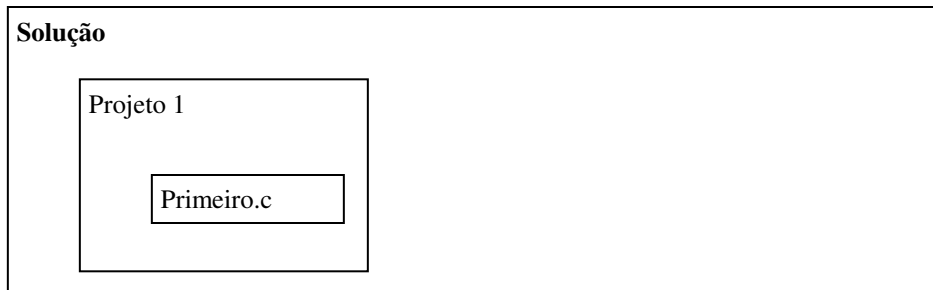


Figura 2 – Hierarquia no Visual Studio dos exemplos deste Curso C

Criar um Novo Projeto

Para criar um novo programa, por exemplo, para testar um exemplo ou resolver um exercício, deve-se criar um novo projeto.

- 1) Após abrir o aplicativo Visual Studio, clique nos seguintes itens no menu: “File” -> “New” -> “Project” (Figura 3).

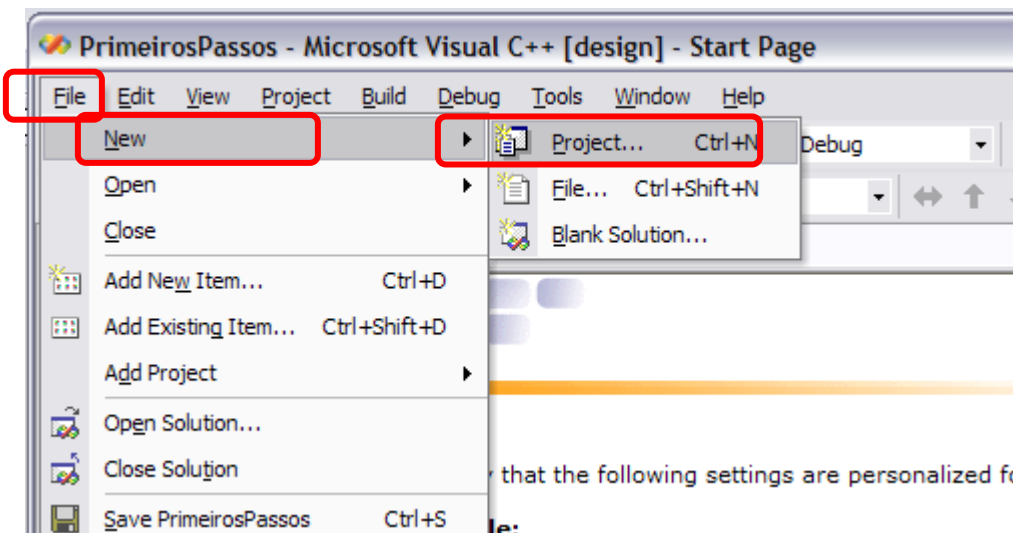


Figura 3 - Criar novo projeto

- 2) Na janela que se abre, selecione “Visual C++ Projects” na lista da metade esquerda. O Visual C não distingue programas em C de programas em C++ (Figura 4).

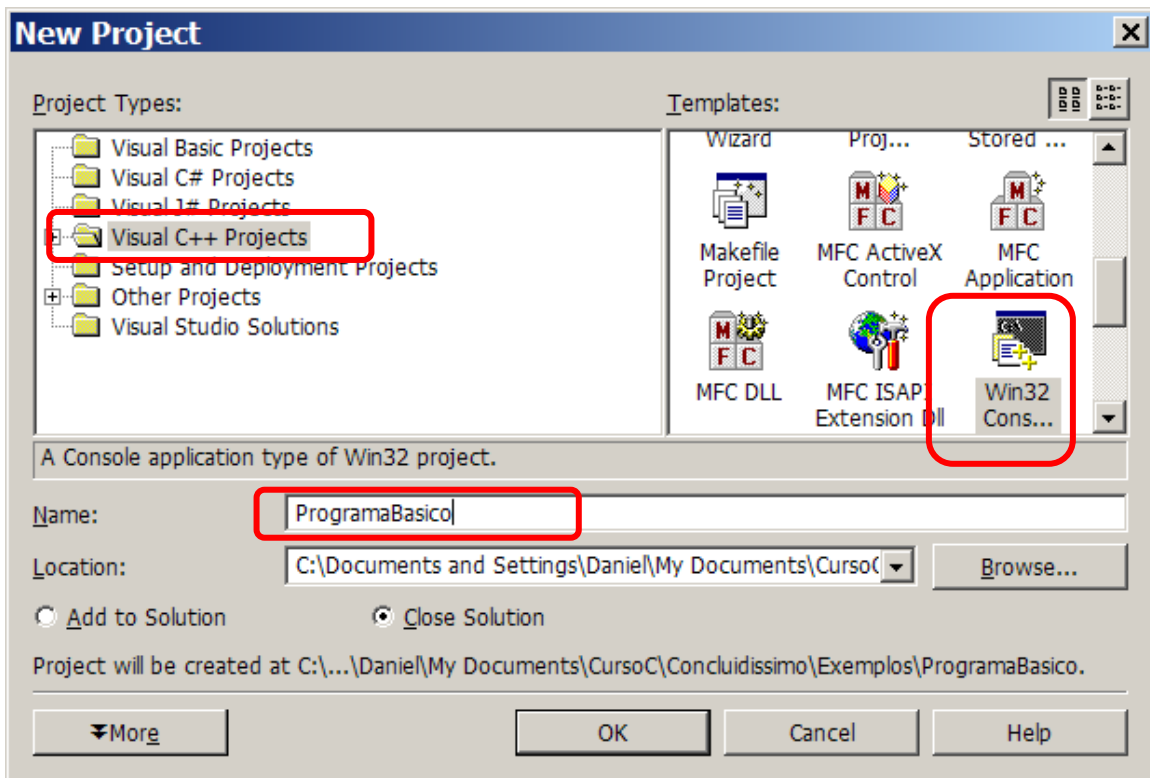


Figura 4 – Selecionar tipo de projeto

- 3) Na metade direita da janela, selecione que tipo de projeto C/C++ deve ser criado (ainda na Figura 4). No momento, interessa apenas uma aplicação simples para ser executada no ambiente Windows (“Win32 Project”).
- 4) Na parte inferior da mesma tela (ainda na Figura 4), escreva o nome do seu projeto e escolha em qual pasta ele deve ser armazenado. O Visual Studio assume automaticamente que deve ser na pasta “Visual Studio Projects”, localizada em “Meus Documentos”. Note que todo o projeto é armazenado como uma nova pasta. Clique em OK.

- 5) A próxima tela tem a função de configurar os principais parâmetros do projeto (Figura 5). Aqui você deve realizar duas alterações importantes para poder compilar os exemplos e exercícios deste curso. Clique na aba “Application Settings” para acessar as configurações. A janela troca de conteúdo. No bloco “Application type” selecione “Console Application”. Ative também a opção “Empty project”. Clique em OK.

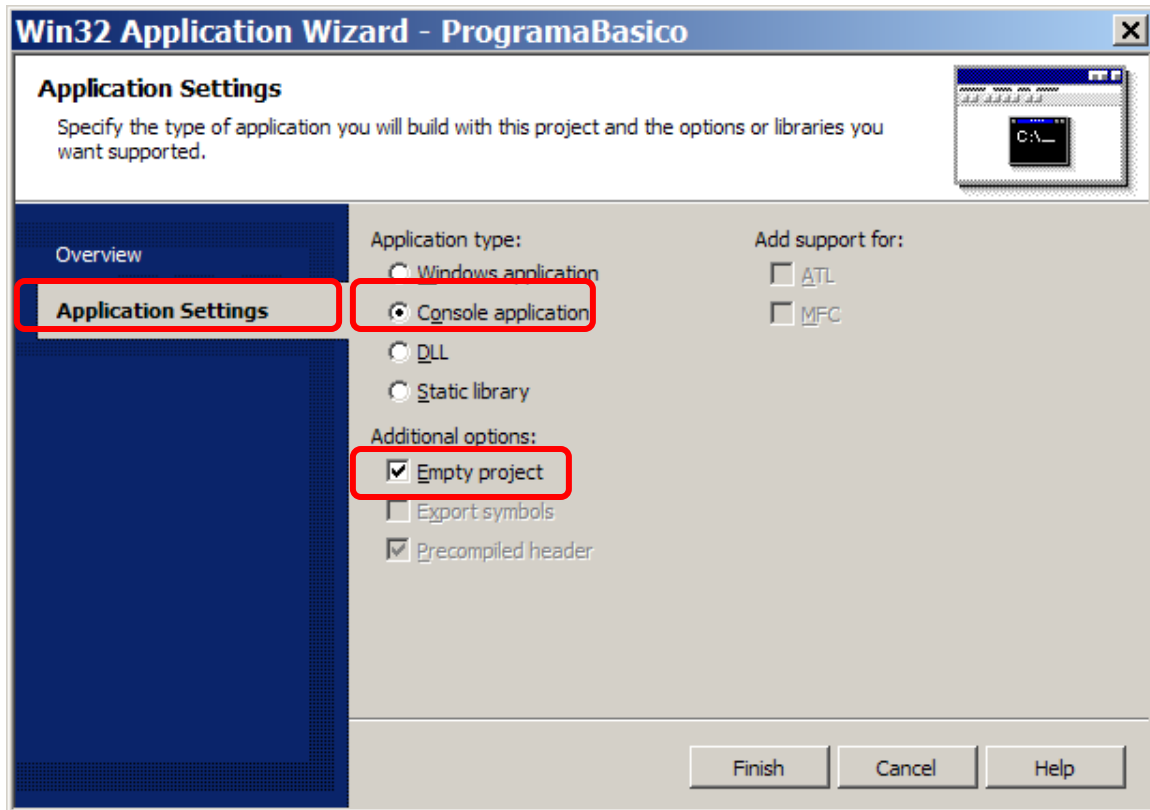


Figura 5 - Configurar o projeto

- 6) O novo projeto não possui nenhum arquivo de código fonte. É necessário adicionar o arquivo manualmente (Figura 6). Na tela principal do Visual Studio, clique com o botão direito do mouse sobre “Source Files” na coluna da direita. No menu que se abre, selecione “Add” -> “Add new item”.

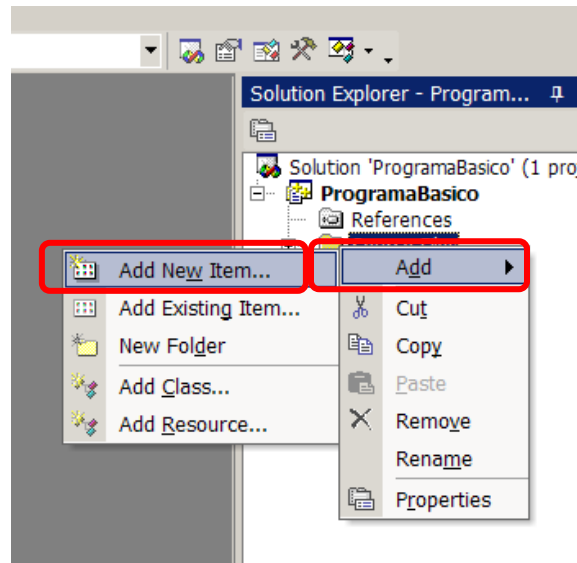


Figura 6 - Adicionar novo arquivo ao projeto

- 7) Na tela que aparece, selecione “C++ File”, mesmo que seu programa seja em C. Indique um nome de arquivo, cuja extensão deve ser .c ao invés de .cpp como sugerido pelo Visual Studio (Figura 7). Clique em OK.

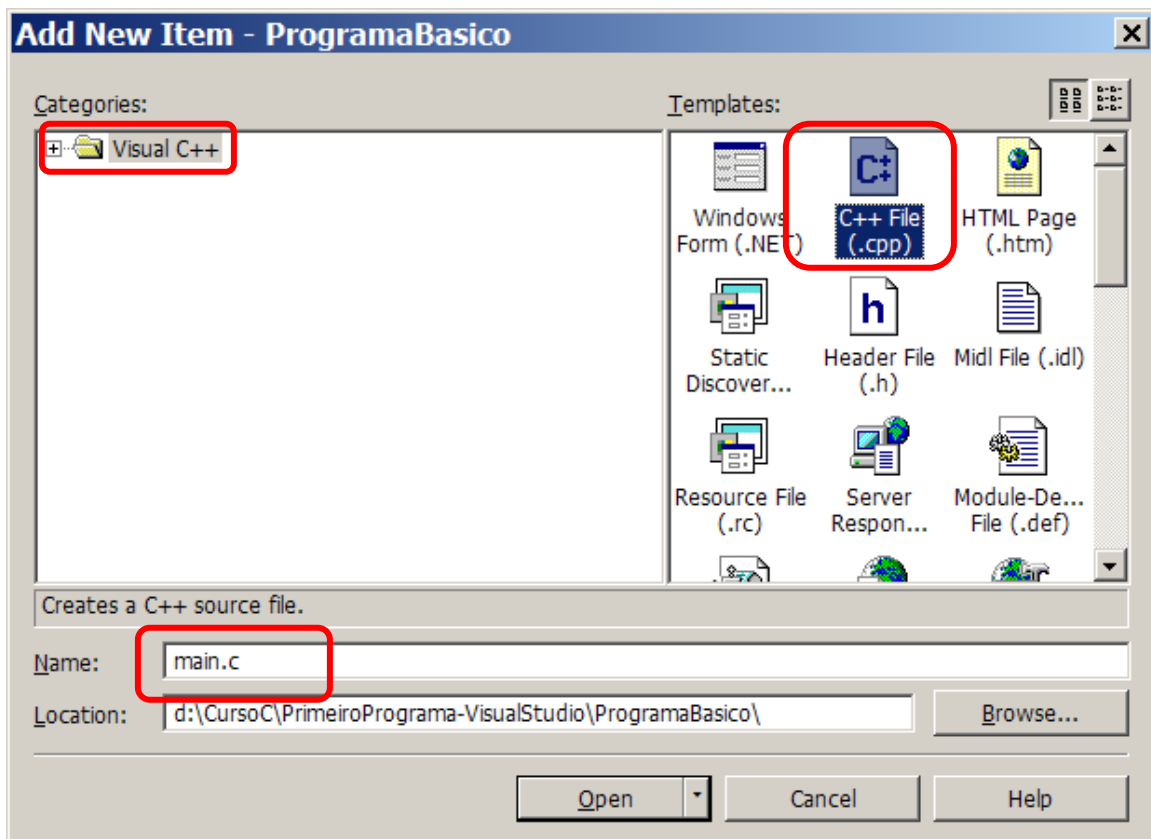


Figura 7 – Selecionar tipo do novo arquivo

- 8) Na tela principal do Visual Studio aparecerá um editor para escrever seu código fonte no arquivo recém criado.

Editar o Código Fonte

Para escrever o código fonte, você precisa primeiro abrir o arquivo que conterá o mesmo. Todos os arquivos do projeto estão organizados no “Solution Explorer”, localizado normalmente do lado direito ou esquerdo do Visual Studio. Clique duas vezes sobre o nome do arquivo principal (“main.c”) para abrir a tela de edição de código fonte (Figura 8).

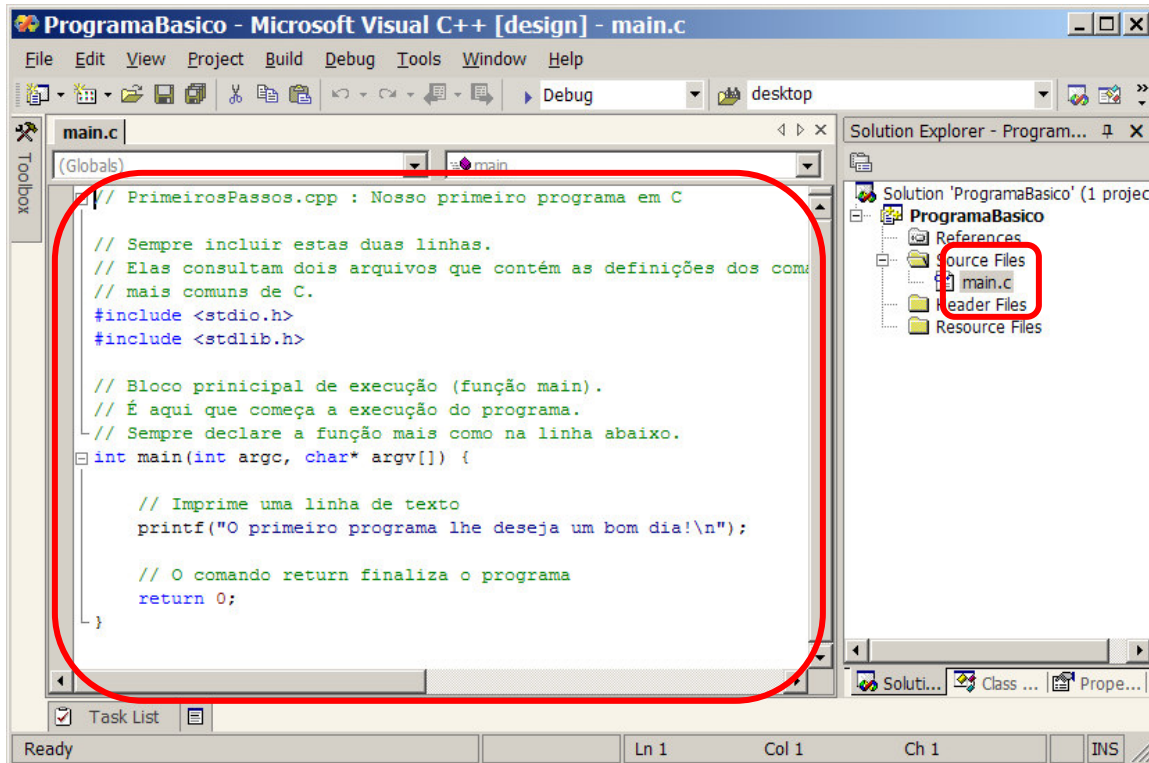


Figura 8 - Editar código fonte

Salvar o Código Fonte

Para salvar seu programa, clique no respectivo ícone na barra de ferramentas, acima do nome do programa, ou no menu selecione “File” -> “Save” (Figura 9).

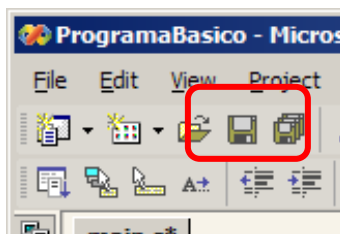


Figura 9 - Salvar

Executar o Programa

Para executar o programa, selecione no menu: “Debug” -> “Start” ou simplesmente pressione F5 (Figura 10). O Visual Studio compilará o seu código fonte e caso não encontre erros, iniciará a execução do seu programa.

Caso existam erros, eles serão listados na parte inferior da janela do Visual Studio. Clicando sobre a mensagem de erro, o editor move o cursor para a linha que gerou a mesma.

Concentre-se somente nos primeiros erros da lista. Você logo perceberá que os demais erros são “consequências” devido ao fato de que o primeiro erro impede o compilador de entender o restante do código fonte.

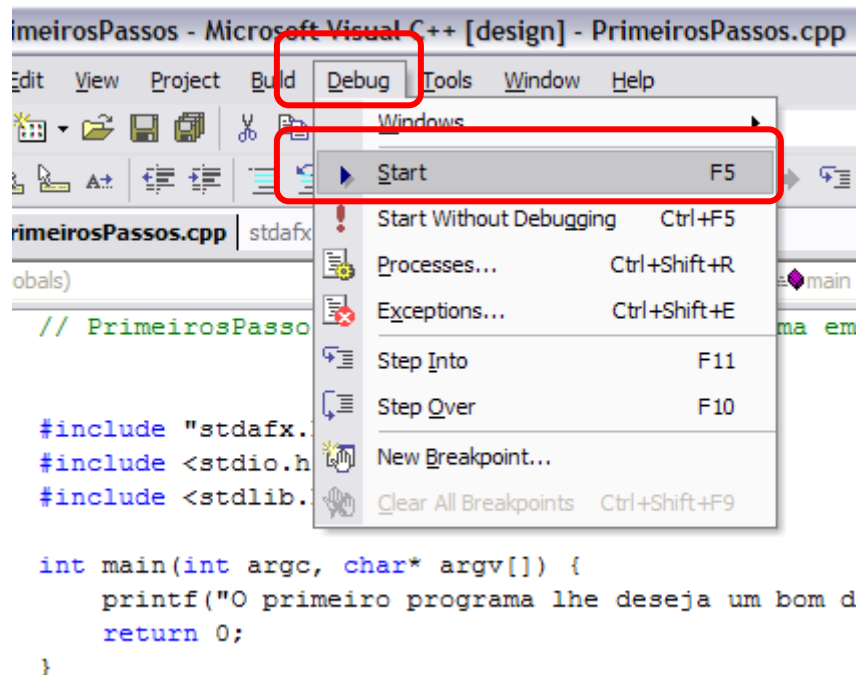


Figura 10 - Executar o programa

Abrir um Projeto

Para abrir um projeto já existente, como os exemplos deste curso, basta clicar duas vezes sobre o arquivo .vcproj do devido projeto (exemplo Figura 11). Ou pelo menu, selecione “File” -> “Open” -> “Project”

Observe que, por padrão, o Visual Studio sempre salva os projetos na pasta “Visual Studio Projects”, localizada em “Meus Documentos”.

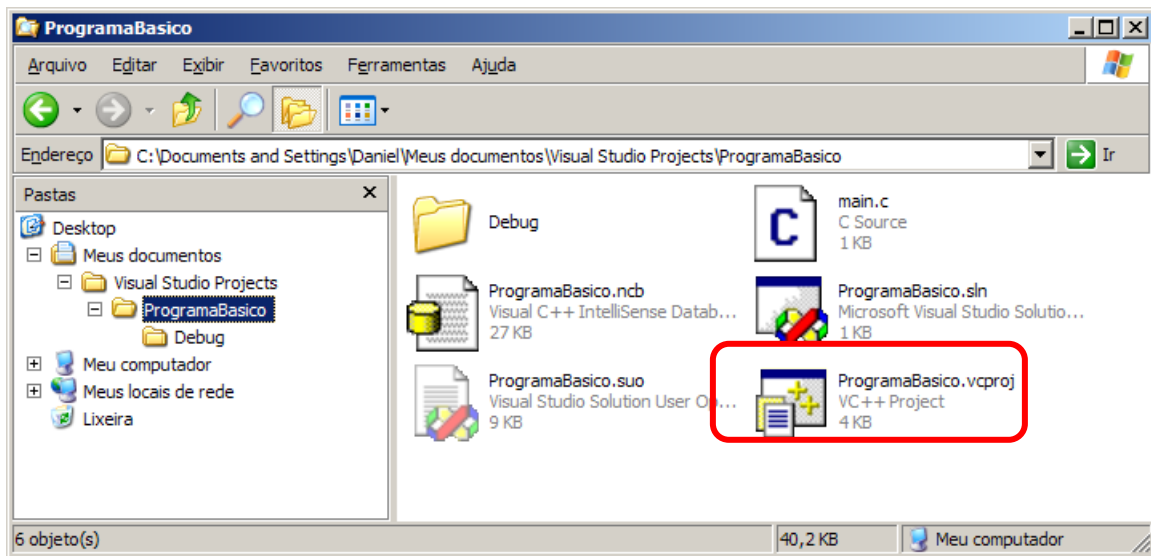


Figura 11 - Abrir projeto pelo Windows