Erros Comuns de Programação

2.1 Esquecer de encerrar um comentário com */.

2.2 Começar um comentário com os caracteres */ ou terminar com /*.

2.3 Em um programa, digitar como **print** o nome da função de saída **printf.**

2.4 Usar uma letra maiúscula onde devia ser usada uma letra minúscula (por exemplo, digitar **Main** em vez de **main).**

2.5 Colocar declarações de variáveis entre instruções executáveis.

2.6 O cálculo de uma instrução de atribuição deve estar no lado direito do operador =. É um erro de sintaxe colocar o cálculo no lado esquerdo de um operador de atribuição.

2.7 Esquecer-se de uma ou ambas as aspas duplas em torno de uma string de controle de formato de **print f** ou **scanf.**

2.8 Em uma especificação de conversão, esquecer-se do % na string de controle de formato de **printf** ou **scanf.**

2.9 Colocar uma seqüência de escape como **\n** fora da string de controle de formato de **printf** ou **scanf.**

2.10 Esquecer-se de incluir em uma instrução **printf** que contém especificadores de conversão as expressões cujos valores devem ser impressos.

2.11 Não fornecer um especificador de conversão para uma instrução **printf,** quando tal é exigido para imprimir uma expressão.

2.12 Colocar, dentro de uma string de controle de formato, a vírgula que deve separar a string de controle de formato das expressões a serem impressas.

2.13 Esquecer-se de preceder uma variável, em uma instrução **scanf**, de um e-comercial quando essa variável deve obrigatoriamente ser precedida por ele.

2.14 Preceder uma variável, incluída em uma instrução **printf,** de um e-comercial quando obrigatoriamente essa variável não deveria ser precedida por ele.

2.15 Normalmente, uma tentativa de dividir por zero não é definida em sistemas computacionais e em geral resulta em um erro fatal, i.e., um erro que faz com que o programa seja encerrado imediatamente sem ter sucesso na realização de sua tarefa. Erros não-fatais permitem que os programas sejam executados até o final, produzindo freqüentemente resultados incorretos.

2.16 Acontecerá um erro de sintaxe se os dois símbolos de qualquer um dos operadores ==, ! =, >= e <= forem separados por espaços.

2.17 Acontecerá um erro de sintaxe se os dois símbolos em qualquer um dos operadores ! =, >= e <= forem invertidos, como em =!, => e =<, respectivamente.

2.18 Confundir o operador de igualdade == com o operador de atribuição =.

2.19 Colocar um ponto-e-vírgula imediatamente à direita do parêntese direito depois de uma condição em uma estrutura **if.**

2.20 Colocar vírgulas (quando não são necessárias) entre os especificadores de conversão na string de controle de formato de uma instrução **scanf.**

Práticas Recomendáveis de Programação

2.1 Todas as funções devem ser precedidas por um comentário descrevendo seu objetivo.

2.2 O último caractere impresso por uma função que realiza qualquer impressão deve ser o de nova linha **(\n).** Isto assegura que a função deixará o cursor da tela posicionado no início de uma nova linha. Procedimentos dessa natureza estimulam a reutilização do software — um objetivo principal em ambientes de desenvolvimento de software.

2.3 Faça o recuo de um nível (três espaços) em todo o texto (corpo) de cada função dentro das chaves que a definem. Isso ressalta a estrutura funcional dos programas e ajuda a torná-los mais fáceis de ler.

2.4 Determine uma convenção para o tamanho de recuo preferido e então aplique-a uniformemente. A tecla de tabulação (tab) pode ser usada para criar recuos, mas as paradas de tabulação podem variar. Recomendamos usar paradas de tabulação de 1/4 de polegada (aproximadamente 6 mm) ou recuar três espaços para cada nível de recuo.

2.5 Embora a inclusão de **<stdio.h>** seja opcional, ela deve ser feita em todos os programas em C que usam funções de entrada/saída da biblioteca padrão. Isto ajuda o compilador a localizar os erros na fase de compilação de seu programa em vez de na fase de execução (quando normalmente os erros são mais difíceis de corrigir).

2.6 Coloque um espaço depois de cada vírgula (,) para tornar o programa mais legível.

2.7 Escolher nomes significativos para as variáveis ajuda a tornar um programa auto-explicativo, i.e., menos comentários se farão necessários.

2.8 A primeira letra de um identificador usado como nome de variável simples deve ser uma letra minúscula. Mais adiante no texto atribuiremos um significado especial aos identificadores que começam com uma letra maiúscula e aos identificadores que usam todas as letras maiúsculas.

2.9 Nomes de variáveis com mais de uma palavra podem ajudar a tornar o programa mais legível. Evite juntar palavras separadas como em **totalpagamentos.** Em vez disso, separe as palavras com sublinhados como em **total_pagamentos** ou, se você desejar juntar as palavras, comece cada palavra depois da primeira com uma letra maiúscula como em **totalPagamentos.**

2.10 Separe as declarações das instruções executáveis em uma função por uma linha em branco,

para ressaltar onde terminam as declarações e começam as instruções.

2.11 Recue as instruções no corpo de uma estrutura **if.**

2.12 Coloque uma linha em branco antes e após todas as estruturas de controle em um programa para melhora sua legibilidade.

2.13 Não deve haver mais de uma instrução por linha em um programa.

2.14 Uma instrução longa pode ser dividida em várias linhas. Se uma instrução deve ocupar mais de uma linha, escolha locais de divisão que façam sentido (como após uma vírgula em uma lista separada por vírgulas). Se uma instrução for dividida em duas ou mais linhas, recue todas as linhas subseqüentes.

2.15 Consulte a tabela de precedência dos operadores ao escrever expressões que contêm muitos operadores. Certifique-se de que os operadores da expressão são executados na ordem adequada. Se você não tiver certeza quanto à ordem de cálculo de uma expressão complexa, use parênteses para impor aquela ordem, exatamente do modo como é feito em expressões algébricas. Não se esqueça de levar em consideração que alguns operadores em C, como o operador de atribuição (=), são associados da direita para a esquerda e não da esquerda para a direita.

Dica de Portabilidade

2.1 Use identificadores com 31 caracteres ou menos. Isso ajuda a assegurar a portabilidade e pode evitar alguns erros sutis de programação.