Coleção Introdução à Linguagem C

Volume 1

Linguagem C

Roteiro de Experimentos para Aulas Práticas

Autores

André Barros de Sales

Georges Daniel Amvame Nze

Organizadora

Márcia Barros de Sales

Coleção Introdução à Linguagem C

Volume 1

Linguagem C

Roteiro de Experimentos para Aulas Práticas

Coleção Introdução à Linguagem C

Volume 1

Linguagem C

Roteiro de Experimentos para Aulas Práticas

Autores

André Barros de Sales

Georges Daniel Amvame Nze

Organizadora

Márcia Barros de Sales

CAD

DEPARTAMENTO DE CIÊNCIAS DA ADMINISTRAÇÃO

Florianópolis

2016

2016 Dos autores

Esta obra é disponibilizada nos termos da Licença Creative Commons Atribuição–NãoComercial–SemDerivações 4.0 Internacional. É permitida a reprodução parcial ou total desta obra, desde que citada a fonte.

Coordenação Gráfica
Denise Aparecida Bunn

Capa, Projeto Gráfico e Diagramação
Claudio José Girardi

Revisão
Claudia Leal Estevão

CORPO EDITORIAL

Alessandra de Linhares Jacobsen
(Universidade Federal de Santa Catarina, Florianópolis, Santa Catarina, Brasil)

Carlos Becker Westphall
(Universidade Federal de Santa Catarina, Florianópolis, Santa Catarina, Brasil)

Cibele Barsalini Martins
(Universidade Federal de Santa Catarina, Florianópolis, Santa Catarina, Brasil)

Iwens Gervasio Sene Junior
(Universidade Federal de Goiás, Goiânia, Goiás, Brasil)

Sérgio Peters
(Universidade Federal de Santa Catarina, Florianópolis, Santa Catarina, Brasil)

Ficha Catalográfica

S163l Sales, André Barros de
Linguagem C [Recurso eletrônico on-line] : roteiro de experimentos para aulas práticas / André Barros de Sales, Georges Daniel Amvame Nze; organizadora Márcia Barros de Sales. – Florianópolis : Departamento de Ciências da Administração/UFSC, 2016.
166p. : il. – (Introdução à Linguagem C; v.1)

ISBN: 978-85-7988-307-1
ISBN COLEÇÃO COMPLETA: 978-85-7988-310-1
Inclui referências
Modo de acesso: http://repositorio.unb.br/handle/10482/21540

1. C (Linguagem de programação de computadores) – Estudo e ensino. I. Nze, Georges Daniel Amvame. II. Sales, Márcia Barros de. III. Título. IV. Série.

CDU: 681.31.06C

Catalogação na publicação por: Onélia Silva Guimarães CRB-14/071

Prefácio

O primeiro ano de uma graduação em Engenharia traz inúmeros desafios aos docentes e estudantes. Um desses desafios é o curso introdutório de programação, que recebe várias denominações e assume vários formatos, mas sempre contendo a difícil tarefa de tornar o estudante capaz de compreender os fundamentos de uma linguagem de programação e de elaborar códigos sintática e semanticamente corretos. E a dificuldade inerente amplia-se quando a linguagem em questão é a linguagem C.

Para o docente, as primeiras funções da Application Programming Interface (API) padrão do C que surgem nos exemplos, printf() e scanf(), trazem consigo dois conceitos avançados da linguagem: funções com argumentos variáveis e ponteiros. Para o iniciante, o símbolo "&" torna-se um mistério: às vezes, ele é necessário (em variáveis inteiras e ponto flutuante, por exemplo); às vezes, não (vetores, strings etc.). Assim, sem o conhecimento e entendimento do conceito de ponteiros fica difícil determinar como usar ou não tal símbolo. Daí, o que era algo preciso e exato para o estudante (um programa de computador!) passa a ser algo aleatório, impreciso

e, naturalmente, complicado. E o professor deve fazer o quê? Explicar ponteiros antes mesmo de estruturas de controle?

Além de fazer uso dessas funções, a linguagem C é, na maioria dos casos, compilada, tendo o estudante de passar por um processo de edição/compilação/execução que é seguido, muitas vezes, de um processo de depuração, cada vez que estiver escrevendo um código. Esse ciclo faz com que o *feedback* entre o que foi escrito e o resultado da execução seja mais demorado e tedioso do que nas linguagens interpretadas, e essas etapas podem gerar novas dúvidas e levar a erros que ampliarão a sensação de dificuldade do estudante.

Como as strings não são tipos primitivos da linguagem, os exemplos práticos escolhidos pelo professor ganham aspectos a serem trabalhados (como o problema do scanf() já citado) que extrapolam o escopo inicial da tarefa. Isso sem contar com as nuances entre conversões de tipos, promoção de tipos em operações aritméticas etc. O docente se vê em um conflito constante entre o que apresentar breve ou detalhadamente e ainda manter o foco, o interesse e o entendimento de seus estudantes.

O presente livro, "Linguagem C: roteiro para experimentos em aulas práticas", escrito pelos professores André e Georges, é fruto da experiência deles acumulada em vários semestres e cujas situações descritas foram apenas poucos exemplos das muitas outras com as quais eles se depararam em sua prática docente. O texto propõe uma série de experimentos a serem trabalhados pelo estudante, individualmente ou sob a supervisão de um instrutor ou do próprio docente. O objetivo é que o estudante se familiarize com a linguagem

e suas principais características, deixando todas as nuances e detalhes para um segundo momento, no qual estará mais seguro em relação à escrita e execução de seus códigos.

É uma iniciativa louvável e que tem mérito na busca de uma alternativa para a complicada tarefa de se ensinar C, uma vez que dá aos instrutores e aos estudantes uma trilha mais suave e segura. Com exemplos completos, gabaritos e exercícios propostos, esta obra certamente servirá de apoio aos estudantes e aos docentes no processo de ensino e aprendizagem dessa complicada, porém necessária e fundamental linguagem.

Edson Alves da Costa Júnior
Professor Adjunto
Curso de Graduação em Engenharia de Software
Faculdade UnB Gama (FGA)
Universidade de Brasília (UnB)

Sumário

Capítulo 12
Strings

Apresentação

Este roteiro de experimentos para aulas práticas de linguagem C é fruto de uma importante iniciativa dos professores e pesquisadores André Barros de Sales e Georges Daniel Amvame Nze que surgiu das demandas dos seus discentes por uma sistematização prática do conteúdo ministrado nas aulas de linguagem C dos cursos de Engenharias da Faculdade UnB Gama (FGA) da Universidade de Brasília (UnB), entre os anos de 2009 e 2013.

Por mais de sete semestres, os professores coletaram informações, por meio de interações nas aulas prestigiadas por mais de 500 alunos e de observações *in loco* no laboratório de informática, sobre os principais problemas encontrados no ensino de programação C.

Este livro é o primeiro de dois volumes da coleção "Introdução à Linguagem C", que foi concebida para oferecer à comunidade acadêmica e aos demais interessados, de forma clara, objetiva e explicativa, um passo a passo de experimentos para aulas práticas, e de exercícios resolvidos, que os auxiliem no aprendizado e na prática da linguagem C. Completa

essa coleção o livro “Linguagem C – Aprendendo com Exercícios Resolvidos”.

A linguagem C foi criada por Dennis Ritchie, na Bell Telephone Laboratories Inc, em 1972, com a finalidade de escrever um sistema operacional utilizando uma linguagem de alto nível (Unix), evitando-se assim uma linguagem de montagem (Assembly). C é uma linguagem de programação compilada, estruturada, imperativa, procedural e de propósito geral, ou seja, pode ser utilizada para desenvolver qualquer tipo de projeto, por exemplo, processamento de registros, sistemas operacionais e outros compiladores de outras linguagens.

Neste roteiro de experimentos, os autores disponibilizam vários exemplos de programas escritos na linguagem C, todos antecedidos de uma breve descrição teórica, um experimento desenvolvido e atividades de fixação baseadas nesses exemplos. É importante frisar que cada experimento é composto de objetivo, orientações, procedimentos e análise.

O livro está organizado da seguinte maneira: os três primeiros capítulos apresentam os elementos básicos da programação em C; os cinco capítulos posteriores apresentam as estruturas de controle de fluxo e de repetição; e os quatro capítulos finais apresentam os conteúdos relacionados a funções, vetores, matrizes e strings.

No aprendizado prático-teórico da linguagem de programação C deste roteiro, os autores recomendam o uso de um compilador para linguagem C como o **MinGW**. Esse compilador pode ser instalado em computador com sistema operacional Windows. Já em computadores com sistema operacional

MinGW

Neste portal, você pode baixar a versão MinG para o Windows, da Microsoft: <https://sourceforge.net/projects/mingw-w64/>. Acesso em: 10 jun. 2016.

Linux, o GNU Compiler Collection (GCC) estará instalado por padrão.

Além do compilador, os autores indicam a utilização de um Ambiente de Desenvolvimento Integrado (IDE) – em inglês, Integrated Development Environment. Para o sistema operacional Windows, alguns ambientes de desenvolvimento gratuitos são: **Editra**, **Notepad++** ou **Dev C++**. O **Editra** e o **Dev C++** também podem ser instalados no Linux, contudo o **Dev C++** pode ser somente instalado para a distribuição alpha. Para o sistema Mac OS X, os autores sugerem o **Editra** ou **Xcode**. Outra alternativa ao IDE é utilizar um editor de texto para digitar o programa e depois compilá-lo em linha de comando. No sistema operacional Ubuntu do Linux, por exemplo, pode-se usar o editor gedit, que destaca as linhas de comando conforme o tipo de arquivo texto, como para arquivos com extensões .c e .cpp, e permite a compilação e execução diretamente em terminal do Ubuntu usando o compilador GCC já instalado.

IDE

Você poderá instalar o IDE apropriado ao sistema operacional do seu computador por meio destes links: Editra, em <http://editra.org/>; Notepad++, em <https://notepad-plus-plus.org/>; Dev C++, em <https://goo.gl/V6IDfM>; e Xcode, em < https://goo.gl/j57nHT >. Acessos em: 14 jun. 2016.

Os autores reforçam que a utilização do IDE é um conhecimento básico. Caso esse conhecimento precise ser reforçado, recomendam a utilização de tutoriais disponíveis no Youtube.

Em tempo, é importante observar que todas as orientações apresentadas nesta obra são sugestões, portanto devem ser analisadas e enriquecidas diante de cada contexto aplicado.

É com grande satisfação que convidamos você a conhecer a primeira obra “Linguagem C – Roteiro de Experimentos para Aulas Práticas” da coleção “Introdução à Linguagem C”.

Organizadora

Márcia de Barros Sales

Capítulo 1

Estrutura Sequencial
Comandos Básicos

Comandos Básicos

Um programa de computador é um conjunto de instruções ou comandos, escritos em uma linguagem de programação, que são executados pelo processador.

Um dos primeiros passos previstos nos fundamentos da aprendizagem da Linguagem C é a apresentação dos conceitos básicos da sua estrutura sequencial .

Os programas com estrutura sequencial possuem instruções, ou comandos, que são executados de forma sequencial, ou seja, um comando depois do outro. Observe os exemplos a seguir:

- Instrução 1: **printf**(“Informe o valor da mercadoria\n”) – a instrução “printf” vai imprimir na tela do computador “Informe o valor da mercadoria”; e o comando “\n” faz com que o cursor passe para a próxima linha depois de a informação ser apresentada pelo printf.
- Instrução 2: **scanf**(“%f”,&valor) – a instrução “scanf” permite a entrada de dados pelo teclado do computador. Esse dado será atribuído a uma posição, ou espaço de memória, normalmente chamado de variável.
- Instrução 3: **printf**(“Papel presente?\n”).
- Instrução 4: **scanf**(“%c”,&presente)

Nesses exemplos, as variáveis são valor e presente.

Experimento 1

Objetivo

O objetivo desse experimento é apresentar a você os conceitos básicos da estrutura sequencial implementada na linguagem de programação C.

Orientações

O experimento que você executará se refere à escrita e definição de um programa que leia o número de alunos e de alunas em uma sala de aula genérica. Você deverá realizar a escrita e alteração do código de modo que ele apresente o número de alunas e alunos, respectivamente (note que é pedida a escrita das alunas e somente depois a dos alunos), na tela do computador, depois da sua execução em IDE. Lembre-se que, para implementar o código do programa referente a todo este roteiro, você precisará utilizar um ambiente integrado de desenvolvimento – Integrated Development Environment (IDE) – como Notepad++ ou Dev-C++.

Procedimentos

Para executar o experimento, siga este passo a passo:

1. Em um sistema operacional, execute o IDE baixado.
2. Digite o código do programa *linguagemC.cpp* (Figura 1.1).
3. Acrescente o seu nome como o aluno do programa na linha 13.
4. Verifique se o código foi digitado corretamente.
5. Compile o programa *linguagemC.cpp*.
6. Caso o programa apresente alguma mensagem de erro, verifique qual linha, qual erro e corrija-o.
7. Caso o programa não apresente nenhuma mensagem de erro, execute-o.
8. Compare a sequência dos comandos apresentados na tela com a sequência dos comandos do programa.

******* Início do Código do programa: *linguagemC.cpp* *******

```c
#include <stdio.h> //Inclusão da Biblioteca para a função de impressão.
#include <stdlib.h> //Inclusão da Biblioteca para a função "system("PAUSE");".

//Bloco main() ou função principal do programa.
int main(){

    //Declaração da variável que armazena o numero de alunos em uma sala.
    int Numalunos;

    //Cabeçalho:
    printf("Universidade de Brasilia \n");
    printf("Disciplina: 113913 -  ICC \n");
    printf("Aluno(a): Meu Nome Completo \n\n");
```

a) Inclusão das bibliotecas, declaração da variável e cabeçalho do programa.

```c
    // Escrevendo mensagem na tela
    printf("\n\n\n Quantos alunos tem na sala? ");

    // Obtendo a informação pelo teclado
    scanf("%d", &Numalunos);
```

b) Mensagem e leitura de dados.

```c
    // Escrevendo o resultado na tela
    printf("\n\n Essa turma possui: %d alunos \n", Numalunos);

    // Parar o sistema para visualizar o resultado
    system("PAUSE");
}
```

c) Finalização do programa.

```
Console
gcc -Wall "C:\icc\linguagemC.cpp" -o "C:\icc\linguagemC.exe"
gcc -Wall "C:\icc\linguagemC.cpp" -o "C:\icc\linguagemC.exe"
Process started >>>
<<< Process finished.
================ READY ================
C++ source file | nb char : 787 | Ln : 27 Col : 1 Sel : 0
```

d) Compilação do programa.

******* Fim código original *******

Figura 1.1 – Código do programa em C: *linguagem.cpp*

Fonte: Elaboração própria

Análise

Vamos comparar a sequência dos comandos apresentados na tela com a sequência dos comandos do programa, conforme figuras 1.1 e 1.2. Observe que cada retorno na linha segue os comandos pré-estabelecidos no programa. A entrada de dados referente à quantidade de alunos é realizada na mesma linha da pergunta: "Quantos alunos tem na sala?". Em seguida, a mensagem: "Press any key to continue ..." aparece na tela para finalizar a visualização da execução do programa.

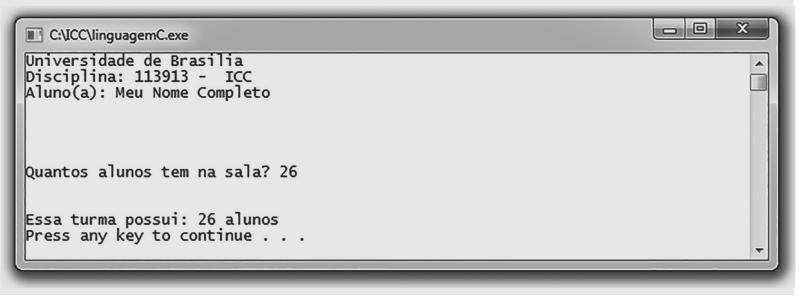


Figura 1.2 – Resultado do código do programa em C: *linguagem.cpp*

Fonte: Elaboração própria

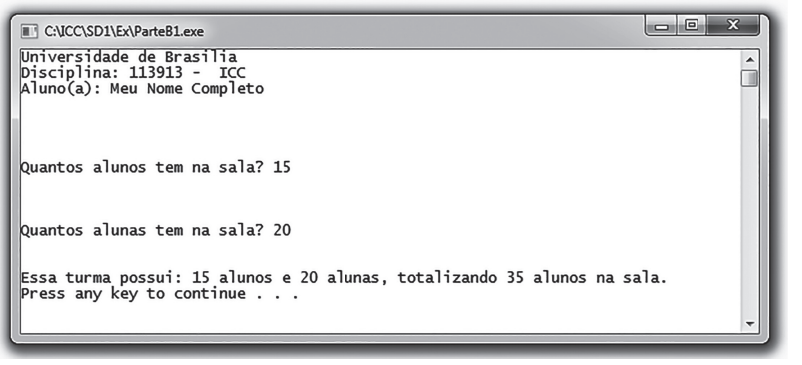
Atividades de Fixação

1. Escreva um programa que leia o número de alunos e de alunas de uma sala. Como saída, o programa deve apresentar o número de alunos, de alunas e o total de alunos na sala.
2. Explique a funcionalidade das instruções ***scanf***, ***printf***, ***\n***, ***;***, ***system("pause")*** e ***//***.
3. Que alteração deve ser feita no exemplo da atividade

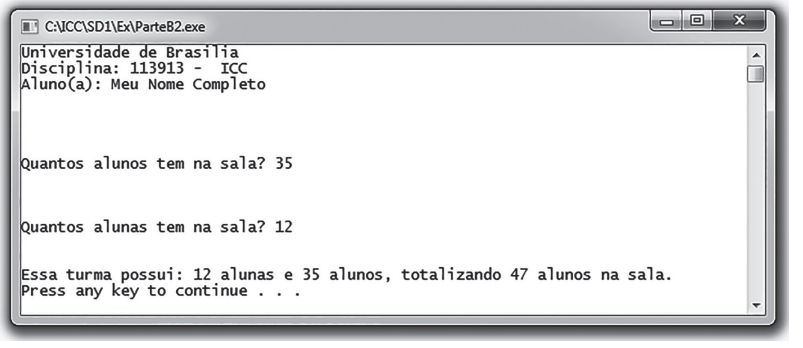
1 para que seja apresentado primeiro o “número de alunas” e, depois, o “número de alunos”? Execute novamente o programa fazendo essas alterações e explique o resultado.

Respostas das Atividades de Fixação

1)



3)



Capítulo 2

Estrutura Sequencial
Tipos de Dados e Variáveis

Tipos de Dados e Variáveis

Os tipos de dados e variáveis são os elementos básicos utilizados em um programa para realizar determinada ação, pré-estabelecida ou não, pelo usuário. As variáveis têm um espaço reservado na memória do computador para armazenar um tipo de dado pré-estabelecido, por um usuário, por exemplo. As variáveis devem receber nomes (nomes de poucos caracteres e de fácil memorização), para serem referenciadas e modificadas quando for necessário pelo programador, ou declarações que especifiquem de que tipo são as variáveis que deverão ser usadas como valor inicial. A seguir, apresentamos os tipos de dados que podem ser usados em um programa básico.

Tipo	Extensão	Escala Numérica em bits
char	8	0 a 255
int	16	-32768 a 32767
float	32	3.4E-38 a 3.4E+38
double	64	1.7E-308 a 1.7E+308
void	0	sem valor

Quadro 2.1 – Exemplos básicos de tipos de dados

Fonte: Elaboração própria

No Quadro 2.1, temos uma representação dos tipos de dados que podem ser utilizados na linguagem C e, a seguir, sua explicação:

- **char (caractere)**: o valor armazenado é um caractere, e caracteres geralmente são armazenados em códigos (usualmente o código ASCII);
- **int:** número inteiro é o tipo padrão e o tamanho do conjunto que pode ser representado e normalmente depende da máquina em que o programa está rodando;
- **float:** número em ponto flutuante de precisão simples é conhecido normalmente como número real;
- **double:** número em ponto flutuante de precisão dupla;
- **void:** esse tipo de dado serve para indicar que um resultado não tem um tipo definido – uma das aplicações desse tipo em C é criar um tipo vazio que pode posteriormente ser modificado para um dos tipos anteriores.

Quanto às variáveis, você poderá usar nomes no código que contenham somente letras e dígitos, separados por um subscrito "_". Mas lembramos que o "_" também será contado como letra se for usado para unir uma palavra. Pela norma da linguagem, todo nome de variável deve ser iniciado por uma letra (lembrando que há diferenciação entre letras maiúsculas e minúsculas nessa linguagem). Antes de inventar qualquer tipo de nome no programa, sugerimos que você não utilize um nome que não faça parte da lista de palavras reservadas pelo compilador C.

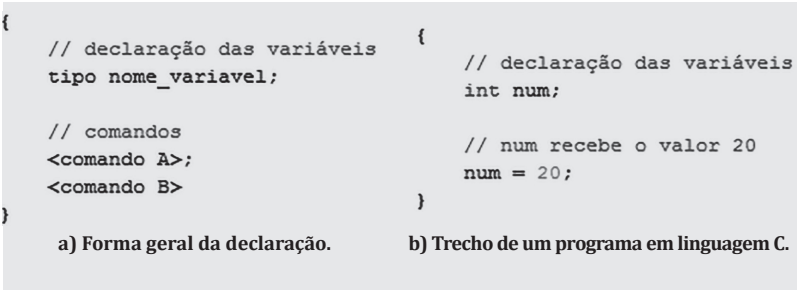


Figura 2.1 – Tipo de dados e variáveis em linguagem C

Fonte: Elaboração própria

Nos itens a) e b) da Figura 2.1, temos a representação de um trecho de programa em sua forma geral e em linguagem C, respectivamente.

Na Figura 2.2, mais à frente, apresentamos um exemplo de programa escrito na linguagem C para representar o uso básico de tipos de dados e variáveis. Esse exemplo é composto de diferentes partes: inclusão das bibliotecas, declaração da variável, mensagem escrita na tela e texto da condição simples.

Experimento 2

Objetivo

O objetivo desse experimento é mostrar a você a seleção simples das estruturas de controle na linguagem de programação C.

Orientações

Nesse experimento, voce deve fazer o programa *Dados-Variaveis.cpp* declarar uma variável cujo nome é chuteira, que, em seguida, recebe um valor aleatório.

Procedimentos

Para executar o experimento, siga este passo a passo:

1. Em um sistema operacional, execute um IDE.
2. Digite o código do programa *DadosVariaveis.cpp* (Figura 2.2).
3. Acrescente o seu nome como o aluno do programa na linha 16.
4. Verifique se o código foi digitado corretamente.
5. Compile o programa *DadosVariaveis.cpp*.
6. Caso o programa apresente alguma mensagem de erro, verifique em qual linha, qual erro e corrija-o.
7. Caso o programa não apresente nenhuma mensagem de erro, execute-o.
8. Compare a sequência dos comandos apresentados na tela com a sequência dos comandos do programa.

******* Início do Código do programa: *DadosVariaveis.cpp* *******

```
1  #include <stdio.h> // Biblioteca de entrada e saida
2  #include <stdlib.h> //Biblioteca para uso do comando system("pause")
3
4
5
6  //Bloco main() ou função principal do programa.
7  int main() {
8
9      // Declaração das variaveis
10     int chuteira; // Variavel chuteira declara como valor inteiro
11     chuteira = 25; // Inicia a quantidade de chuteira com um total de 25
12
13     // Cabeçalho
14     printf("Universidade de Brasilia \n");
15     printf("Disciplina: 113913 -  ICC \n");
16     printf("Aluno(a): Meu Nome Completo \n\n");
17
```

a) Inclusão das bibliotecas, declaração da variável e cabeçalho do programa.

```
18         //escrevendo mensagem na tela
19         printf("\nA quantidade de chuteira em estoque sera:");
20         //imprima o resultado
21         printf("\n %d \n\n", chuteira);
22
```

b) Mensagem e leitura de dados.

```
22
23     //para o sistema para visualiar o resultado
24     system("PAUSE");
25     return 0;
26 }
```

c) Finalização do programa.

```
Console
gcc -Wall "C:\jcc\DadosVariaveis.cpp" -o "C:\jcc\DadosVariaveis.exe"
gcc -Wall "C:\jcc\DadosVariaveis.cpp" -o "C:\jcc\DadosVariaveis.exe"
Process started >>>
<<< Process finished.
================ READY ================
```

d) Compilação do programa.

******* Fim código original *******

Figura 2.2 – Código do programa em C: *DadosVariaveis.cpp*

Fonte: Elaboração própria

Análise

Compare a sequência dos comandos apresentados na tela com a sequência dos comandos do programa anterior. Perceba que cada retorno na linha segue os comandos pré-estabelecidos no programa. A entrada de dados pré-estabelecida à quantidade de chuteiras é armazenada em espaço de memória e somente será mostrada na tela quando o usuário fizer sua chamada. Depois da visualização da mensagem "A quantidade de chuteiras em estoque sera:", aparece o valor já declarado da quantidade de chuteiras, que é 25. Em seguida, a mensagem "Pressione qualquer tecla para continuar..." finaliza a visualização da execução do programa.

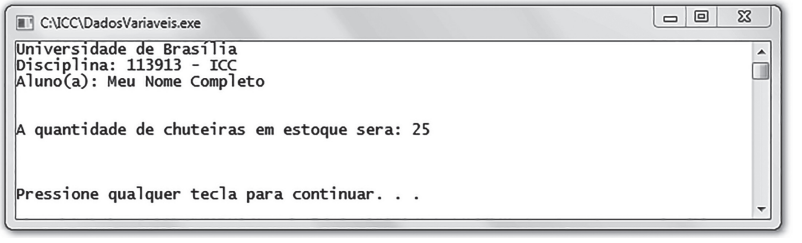


Figura 2.3 – Resultado do código do programa em C: *linguagem.cpp*

Fonte: Elaboração própria

Atividades de Fixação

1. Escreva um programa que leia o número de chuteiras marca A, marca B e marca C de uma loja de esporte. Os valores terão de ser entrados via teclado. Como saída, o programa deve apresentar o número de chuteiras discriminadas por marca (marca A, marca B e marca C).

Utilizando o caráter especial **\t** dentro do *printf*, desejamos que você obtenha uma saída parecida com esta:

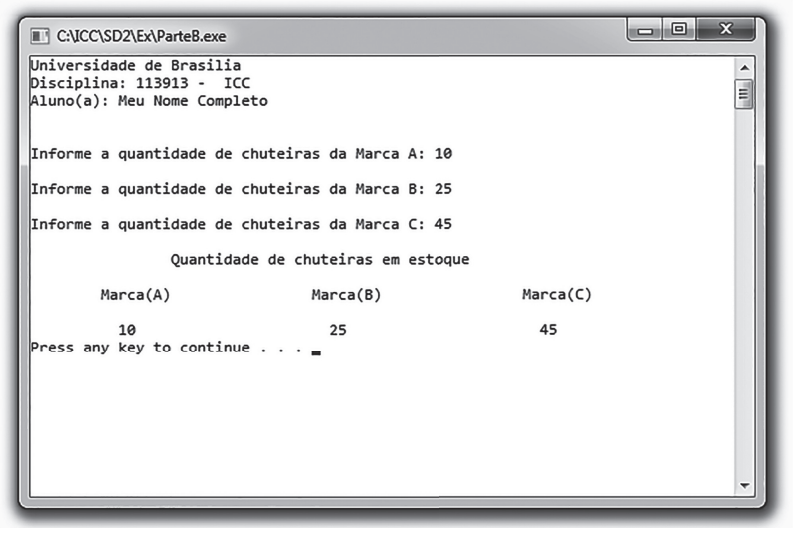
Quantidade de chuteiras em estoque

Marca (A)	Marca (B)	Marca (C)
10	25	45

2. Explique a funcionalidade de todos os outros tipos de dados e variáveis encontrados na Linguagem C.

Respostas das Atividades de Fixação

1)



Capítulo 3

Estrutura Sequencial
Variáveis e Operadores

Variáveis e Operadores

> ***Repositórios on-line***
>
> Consulte alguns exemplos nestes repositórios *on-line*: A Little C Primer/C Operators, disponível em: <https://en.wikibooks.org/wiki/A_Little_C_Primer/C_Operators>; The C Book, disponível em: <http://publications.gbdirect.co.uk/c_book/>; e Operators and Punctuators, disponível em <http://tigcc.ticalc.org/doc/opers.html>. Acessos em: 17 fev. 2017.

Existem vários tipos de operadores na linguagem C, como de atribuição, aritmético, de endereçamento, cuja definição e exemplos podem ser encontrados em **repositórios *on-line***. As expressões que decorrem da linguagem C devem ser escritas de maneira legível e na mesma linha. Caso as expressões sejam separadas, pode haver um problema de pré-processamento na hora de identificar a operação que você deseja realizar. O uso dos parênteses, por exemplo, faz com que a ordem das operações a serem realizadas em determinado cálculo ou comparação seja alterada. Dessa forma, as operações escritas entre parênteses devem ser executadas antes das que estejam fora deles. Nesse caso, as expressões que estiverem entre parênteses serão avaliadas primeiro, ou seja, para fora, dependendo do número de parênteses presentes na linha de operação. No Quadro 3.1, temos um exemplo de operadores aritméticos que podem ser usados num programa básico.

Operador	Função	Descrição
*	**Multiplicação**	Realiza a operação de multiplicação entre dois ou mais valores.
/	**Divisão**	Realiza a operação de divisão entre dois ou mais valores.
%	**Resto**	Retorna o resto da divisão de dois números.
+	**Adição**	Realiza a operação entre dois ou mais valores.
-	**Subtração**	Realiza a operação de subtração entre dois ou mais valores.

Quadro 3.1 – Exemplos básicos de operadores aritméticos
Fonte: Elaboração própria

Na Figura 3.1, temos uma representação dos tipos de operadores aritméticos que podem ser utilizados na linguagem C. Como vimos no Capítulo 2, no que se refere ao simples uso das variáveis, os nomes que você utilizar no código poderão conter somente letras e dígitos, separados por um subscrito "_", que também será contado como letra se você usá-lo para unir uma palavra. Por norma, todo caractere deve iniciar por uma letra (lembrando que há diferenciação entre letras maiúsculas e minúsculas). Sugerimos que, antes de inventar quaisquer tipos de nome para variáveis no programa, que ele não faça parte da lista de palavras reservadas pelo compilador C.

```
{
    // declaração das variáveis
    tipo nome_variavel;

    // comandos
    <nome_variavel> = <nome_variavel> <operador> <expressao>;
    <nome_variavel> <operador>= <expressao>;
}
```

a) Forma geral da declaração.

```
{
    // declaração da variável
    int tempo=0;

    // possíveis comandos
    tempo = tempo + 20;
    tempo += 20;
}
```

b) Trecho de um programa em linguagem C.

Figura 3.1 – Tipo de operador aritmético em linguagem C

Fonte: Elaboração própria

Nos itens a) e b) da Figura 3.1, temos a representação de um trecho de programa em sua forma geral e em linguagem C, respectivamente.

No experimento 3, a Figura 3.2 apresenta um exemplo de programa escrito na linguagem C para representar o uso básico de tipos de operadores e variáveis. Esse exemplo é composto de diferentes partes: inclusão das bibliotecas, declaração da variável, mensagem escrita na tela e texto da condição simples.

Experimento 3

Objetivo

O objetivo desse experimento é apresentar a você a seleção simples das estruturas de controle codificado na linguagem de programação C.

Orientações

Nesse experimento, você deve fazer o programa *Operadores.cpp* declarar uma variável cujo nome é chuteira e que, em seguida, recebe um valor aleatório.

Procedimentos

Para executar o experimento, siga este passo a passo:

1. Em um sistema operacional, execute um IDE.
2. Digite o código do programa *Operadores.cpp* (Figura 3.2).
3. Acrescente o seu nome como o aluno do programa na linha 19.
4. Verifique se o código foi digitado corretamente.

5. Compile o programa *Operadores.cpp*.
6. Caso o programa apresente alguma mensagem de erro, verifique em qual linha, qual erro e corrija-o.
7. Caso o programa não apresente nenhuma mensagem de erro, execute-o.
8. Compare a sequência dos comandos apresentados na tela com a sequência dos comandos do programa.

******* Início do Código do programa: *Operadores.cpp* *******

```
1  #include <stdio.h> //Inclusão da Biblioteca para a função de impressão.
2  #include <stdlib.h> //Inclusãc da Biblioteca para a função "system("PAUSE");".
3
4  //Bloco main ou função principal do programa.
5  int main(){
6
7      //Declaração da variável que armazenará o número de chuteiras.
8      int chuteira;
9      float valorchut, i, t;//Declaração das variáveis que armazenarão o
10                           //valor das chuteiras(valorchuteira), imposto
11                           //scbre as chuteiras (i) e o total dos impostos (t).
12
13     chuteira = 25; //Atribuição do valor 25 à variável chuteira.
14     valorchut = 100.0; //Atribuição do valor 100 à variável valorchut.
15
16     //Cabeçalho:
17     printf("Universidade de Brasilia \n");
18     printf("Disciplina: 113913 -  ICC \n");
19     printf("Aluno(a): Meu Nome Completo \n\n");
20
```

a) Inclusão das bibliotecas, declaração da variável e cabeçalho do programa.

```
22         //escrevendo mensagem na tela
23         printf("\nA quantidade de chuteira em estoque sera:");
24         //imprima o resultado
25         printf("\n %d \n\n", chuteira);
26         //escrevendo mensagem na tela do novo valor da chuteira
27         printf("\nO valor da chuteira sem imposto sera: R$ %0.2f \n", valorchut);
28
29         //escrevendo mensagem na tela o pedido do valor do imposto
30         printf("\nFavor entrar o valor atual do imposto:\n");
31         //recebendo do teclado o valor do imposto (ex: 0.75)
32         scanf("%f", &i);
33         //escrevendo mensagem na tela do novo valor acrescido do emposto
34         t = valorchut*i + valorchut;
35         printf("\nO valor da chuteira com imposto sera: R$ %0.2f \n\n", t);
```

b) Mensagens e leitura de dados.

```
36
37          //para o sistema para visualiar o resultado
38          system("PAUSE");
39          return 0;
40      }
```

c) Finalização do programa.

```
Console
gcc -Wall C:\jcc\Operadores.cpp -o C:\jcc\Operadores.exe
gcc -Wall C:\jcc\Operadores.cpp -o C:\jcc\Operadores.exe
Process started >>>
<<< Process finished.
================ READY ================
```

d) Compilação do programa.

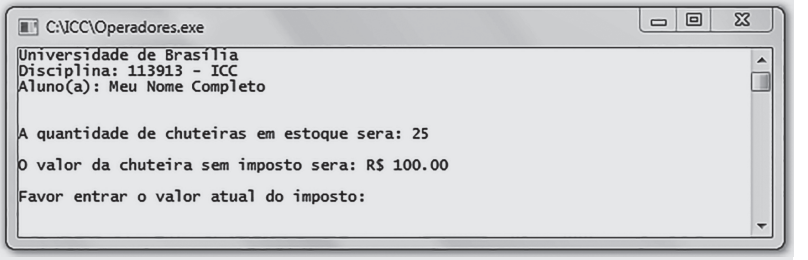
******* Fim código original *******

Figura 3.2 – Código do programa em C: *Operadores.cpp*

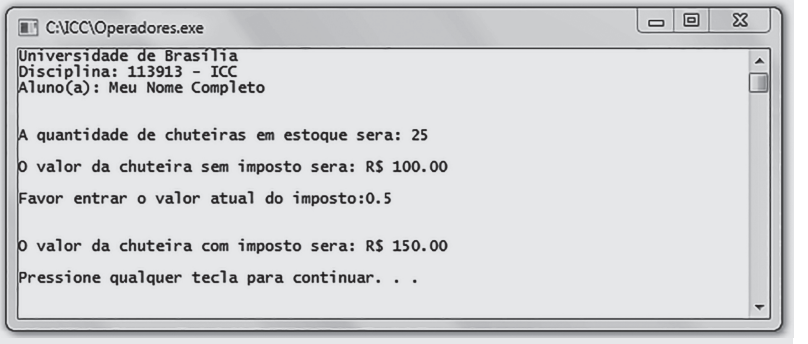
Fonte: Elaboração própria

Análise

Compare a sequência dos comandos apresentados na tela com a sequência dos comandos do programa anterior. Perceba que cada retorno na linha segue os comandos pré-estabelecidos no programa. A entrada de dados pré-estabelecida à quantidade de chuteiras é armazenada em espaço de memória e somente será mostrada na tela quando o usuário fizer sua chamada. Depois da visualização da mensagem: "A quantidade de chuteira em estoque sera:", aparece o valor já declarado da quantidade de chuteiras, ou seja 25. Em seguida, a mensagem "Favor entrar o valor atual do imposto:" surge na tela para receber um valor digitado pelo usuário.



a) Pedido entrada valor imposto.



b) Exibição resultado do programa.

Figura 3.3 – Resultado do código do programa em C: *Operadores.cpp*

Fonte: Elaboração própria

Atividades de Fixação

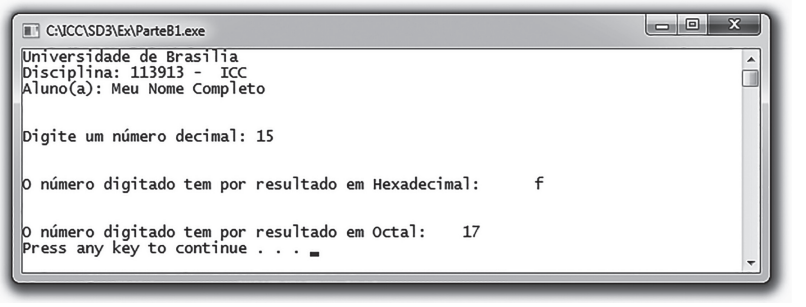
1. Escreva um programa que receba um número decimal e o converta para hexadecimal e octal respectivamente.

2. Com base no programa anterior, escreva um programa que converta 4 *bits* (por exemplo, 0101) de informação binária em decimal.

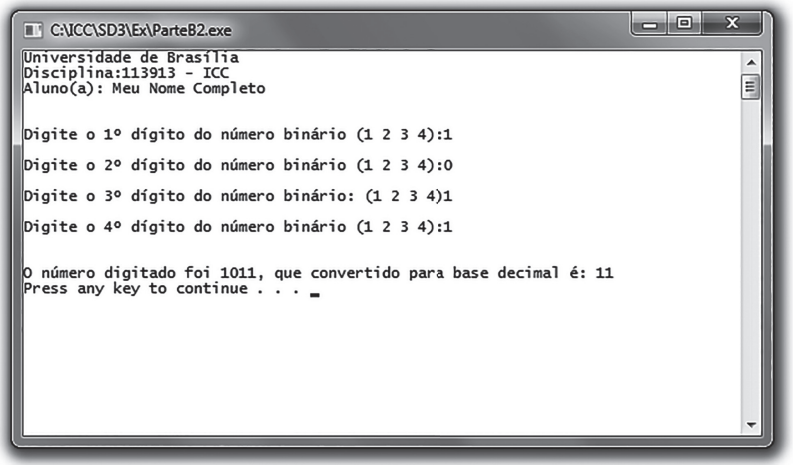
Sabemos da biblioteca padrão "*stdio.h*" que o uso do ***%d*** no *printf( )* permite a escrita de uma variável de entrada (recebida como entrada via teclado ou não) num formato de saída decimal. Dessa forma, podemos pensar em ***%x*** e ***%o*** para formatos em hexadecimal e octal, respectivamente. Já a conversão em binário demanda mais trabalho.

Respostas das Atividades de Fixação

1)



2)



Capítulo 4

Estrutura de Controle de Fluxo
Seleção Simples

Seleção Simples

A estrutura sequencial de comandos é muito utilizada em alguns tipos de programas e significa que todos os comandos de programação são executados em uma sequência. Em certos casos, ou condições, alguns comandos do programa devem ser executados e outros não, devendo-se utilizar a estrutura de controle de fluxo para verificar uma determinada condição para executar determinado comando ou conjunto de comandos. Essa estrutura de controle de fluxo pode ser uma seleção simples ou composta. A seleção simples é uma estrutura de controle que permite, em determinadas condições, executar ou não um trecho do programa. Essa estrutura de controle é muito utilizada nos diversos tipos de linguagens de programação.

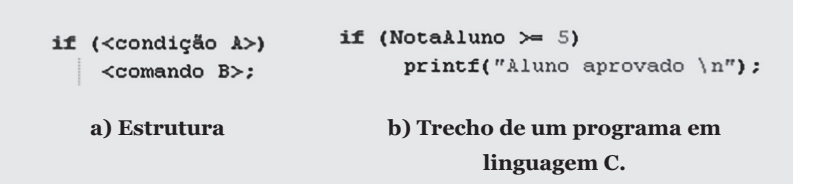


Figura 4.1 – Exemplos de estrutura de controle: condicional simples

Fonte: Elaboração própria

No exemplo a) da Figura 4.1, o **<comando B>** somente será executado se a **<condição A>** for verdadeira. Se o resultado da **<condição A>** for falso, o **<comando B>** não será executado e o programa continuará depois do ponto e vírgula (**;**) do **<comando B>**. O resultado da **<condição A>** deve ser um valor lógico.

No exemplo b) da Figura 4.1, o comando **printf("Aluno aprovado")** somente será executado caso o valor da variável **nota** seja maior ou igual ao valor 5.

Para executar mais de um comando depois de uma condição, é necessário que esse conjunto de comandos fique dentro de um bloco. Na linguagem C, um bloco é representado pelos comandos contidos dentro de chaves **{ }**. O item a) da Figura 4.2 apresenta um trecho de programa em que mais de um comando será executado caso a **<condição A>** seja verdadeira.

```
if (condição A)
    {
        <comando A>;
        <comando B>;
        <.........>;
    }
```

a) Estrutura.

```
if (nota > 9)
    {
        printf("Aluno aprovado.");
        printf("Parabens!");
    }
```

b) Trecho de um programa em linguagem C.

Figura 4.2 – Estrutura de Controle: condicional simples

Fonte: Elaboração própria

O item b) da Figura 4.2 apresenta um trecho de programa em que algumas mensagens serão escritas na tela caso o valor da variável nota seja superior a 9.

Na Figura 4.3, temos um exemplo de programa escrito na linguagem C para representar uma estrutura de seleção usando uma sequência condicional simples. Esse exemplo é composto de diferentes partes: inclusão das bibliotecas, declaração de variável, mensagem escrita na tela e o texto da condição simples.

```c
#include <stdio.h> // Biblioteca de entrada e saida
#include <stdlib.h> //Biblioteca para uso do comando system("pause")

int main() {

    // Declaração das variáveis
    int TtlAlunos; // Quantidades totais de alunos e alunas

    //Cabeçalho
    printf("Universidade de Brasilia \n");
    printf("Disciplina: 113913 -  ICC \n");
    printf("Aluno(a): Meu Nome Completo \n\n");

    // Escrevendo mensagem na tela
    printf("\n\n\n Informe a quantidade total de alunos da sala");
    // Obtendo a informação pelo teclado
    scanf("%d", &TtlAlunos);

    // Teste para verificar  se quantidade de alunos é maior que zero
    if (TtlAlunos>0)
    {
    // Imprima o resultado
    printf("\n Essa turma possui alunos. \n\n");
    printf("Essa turma possui um total de %d alunos.\n\n", TtlAlunos);
    }
    // Para o sistema visualizar o resultado
    system("PAUSE");
    return 0;
}
```

Figura 4.3 – Programa exemplo de uma condicional simples na linguagem C

Fonte: Elaboração própria

No exemplo *ExEstCSimples.cpp*, quando o valor da variável *TtlAlunos* for maior que zero, aparecerá na tela a mensagem que a turma possui *TtlAlunos* alunos. Note que *TtlAlunos* é o número total de alunos. Caso a quantidade de alunos seja menor que zero, nenhuma mensagem aparecerá na tela.

Experimento 4

Objetivo

O objetivo desse experimento é fazê-lo praticar a seleção simples das estruturas de controle codificado na linguagem de programação C.

Orientações

Nesse experimento, você deve fazer o programa *EstruCondSimples1.cpp* ler o número de alunas e, depois, de alunos numa sala de aula genérica. A seleção simples é utilizada para verificar se a sala de aula possui a mesma quantidade de alunos e alunas ou se possui mais alunas do que alunos. Caso uma dessas verificações seja verdadeira, o programa escreve uma mensagem específica.

Procedimentos

Para executar o experimento, siga este passo a passo respeitando a sequência dada nas partes A e B:

Parte A

1. Em um sistema operacional, execute um IDE.
2. Digite o código do programa *EstruCondSimples1.cpp,* da Figura 4.3.
3. Acrescente o seu nome como o aluno do programa, na linha 12.
4. Verifique se o código foi digitado corretamente.
5. Compile o programa *EstruCondSimples1.cpp*.
6. Caso o programa apresente alguma mensagem de erro, verifique em qual linha, qual o erro e corrija-o.

7. Caso o programa não apresente nenhuma mensagem de erro, execute o programa.
8. Compare a sequência dos comandos apresentados na tela com a dos comandos do programa.

Parte B

1. Escreva um programa *EstruCondSimplesA.cpp* que leia a quantidade de alunas e alunos (Figura 4.4).
2. Verifique se o programa informou se essa turma possui mais alunos ou mais alunas. Se essa turma possuir mais alunas do que alunos, informe o total dos alunos dessa turma. Esse programa deve apresentar o resultado conforme a Figura 4.5.
3. Verifique se o código foi digitado corretamente.
4. Compile o programa *EstruCondSimplesA.cpp*.
5. Caso apresente alguma mensagem de erro, identifique a linha e o erro e corrija-o.
6. Não apresentando nenhuma mensagem de erro, execute o programa.

***** Início do Código do programa: *EstruCondSimples1.cpp* *****

```
1   #include <stdio.h> // Biblioteca de entrada e saida
2   #include <stdlib.h> //Biblioteca para uso do comando system("pause")
3
4   int main() {
5
6       // Declaração das variáveis
7       int QtdAlunos; // Quantidade de alunos
8       int QtdAlunas; // Quantidade de alunas
9       int TtlAlunos; // Quantidades total de alunos e alunas
10
11      //Cabeçalho
12      printf("Universidade de Brasilia \n");
13      printf("Disciplina: 113913 -  ICC \n");
14      printf("Aluno(a): Meu Nome Completo \n\n");
```

a) Inclusão das bibliotecas, declaração das variáveis e cabeçalho do programa.

```
16      // Escrevendo mensagem na tela
17      printf("\n\n\n Informe a quantidade de alunos da sala?");
18      // Obtendo a informação pelo teclado
19      scanf("%d", &QtdAlunos);
20      // Escrevendo mensagem na tela
21      printf("\n\n\n Informe a quantidade de alunas da sala?");
22      // Obtendo a informação pelo teclado
23      scanf("%d", &QtdAlunas);
24
25      // Cálculo do total de alunos
26      TtlAlunos = QtdAlunos + QtdAlunas;
27
```

b) Mensagens e leitura de dados.

```
28      // Teste para verificar  se a quantidade de alunos é igual à de alunas
29      if (QtdAlunas==QtdAlunos)
30      // Imprimindo o resultado
31      printf("\n Essa turma possui a mesma quantidade de alunos e alunas. \n\n");
32
33      // Teste para verificar  se a quantidade de alunas é maior que a de alunos
34      if (QtdAlunas>QtdAlunos)
35      {
36      // Imprima o resultado
37      printf("\n Essa turma possui mais alunas que alunos. \n");
38      printf(" Essa turma possui: %d alunos.\n\n", TtlAlunos);
39      }
40
41      // Para o sistema visualizar o resultado
42      system("PAUSE");
43      return 0;
44  }
```

c) Condicional Simples.

```
Console
================ READY ================
gcc -Wall c:\icc\EstrucondSimples1.cpp -o c:\icc\EstrucondSimples1.exe
gcc -Wall c:\icc\EstrucondSimples1.cpp -o c:\icc\EstrucondSimples1.exe
Process started >>>
<<< Process finished.
================ READY ================
C++ source file | nb char : 1903 | Ln : 1  Col : 1  Sel : 0 | Dos\Windows | ANSI | INS
```

d) Compilação do programa.

******* Fim código original *******

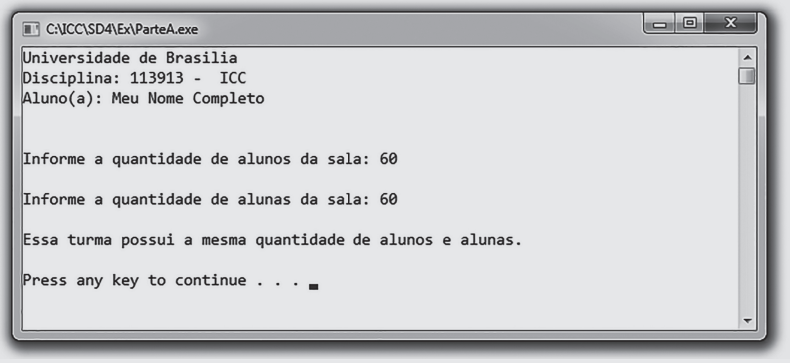
Figura 4.4 – Código do programa em C: *EstruCondSimples1.cpp*

Fonte: Elaboração própria

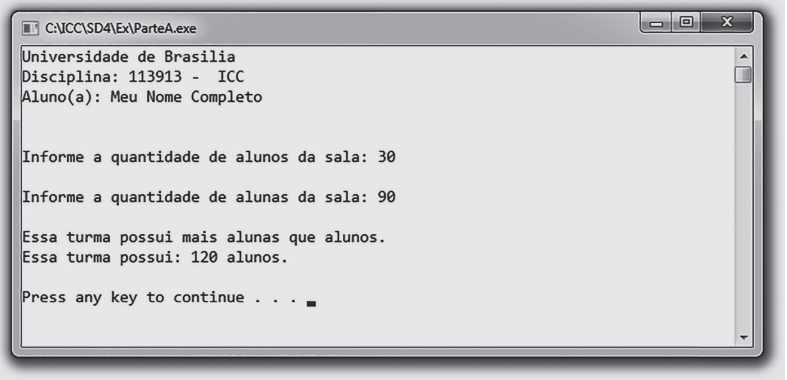
Análise

Apresentamos o resultado da execução do procedimento B do programa *EstruCondSimples1.cpp* na Figura 4.5. O pro-

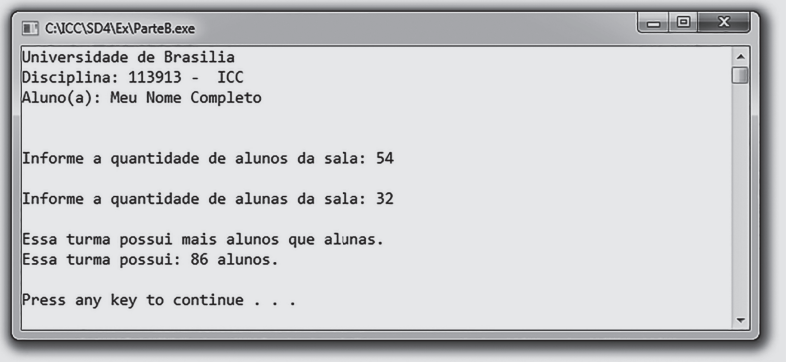
grama apresenta um cabeçalho, em seguida solicita a quantidade de alunos e, depois, de alunas numa sala de aula. Depois, o programa informa se essa sala de aula possui a mesma quantidade de alunos e alunas, se a turma possui mais alunas do que alunos ou se possui mais alunos do que alunas, informando o total de estudantes.



a) Condição de quantidades iguais de alunos e alunas.



b) Condição em que há mais alunas do que alunos.



c) Condição em que há mais alunos do que alunas.

Figura 4.5 – Resultado do código do programa em C: *EstruCondSimples1.cpp*

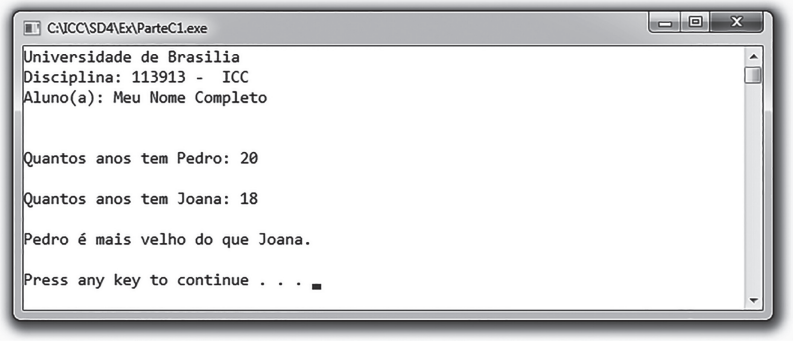
Fonte: Elaboração própria

Atividades de Fixação

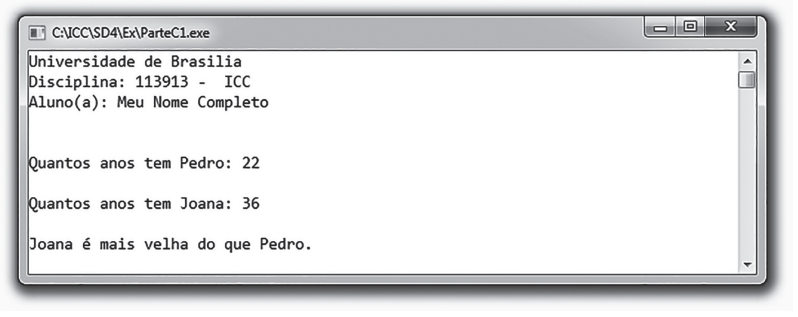
1. Desenvolva um programa para comparar as idades de Pedro e Joana e informar quem é o mais velho. Dados de entrada: idade do Pedro e da Joana (tipo das variáveis: inteiro, valor em anos). Observação: essas pessoas possuem idades diferentes.
2. Desenvolva um programa para calcular e comparar a área de dois retângulos A e B; ele deverá dizer qual retângulo possui a maior área ou se eles possuem tamanhos iguais. Dados de entrada: tamanho da base e da altura (tipo das variáveis: inteiro, valor em centímetros).

Respostas das Atividades de Fixação

1)

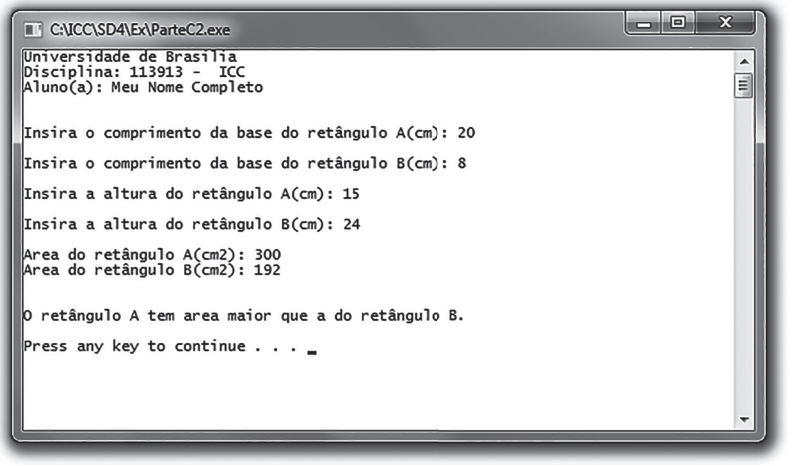


a) Resposta para a condição de Pedro ser mais velho do que Joana.



b) Resposta para a condição de Joana ser mais velha do que Pedro.

2)



c) Resposta para a condição de a área do retângulo A ser maior do que a do retângulo B.

Capítulo 5

Estrutura de Controle de Fluxo Condicional Composta

Condicional Composta

A estrutura condicional composta possibilita que uma ação ou bloco de ações somente seja executada se uma condição for satisfeita e outra ação ou bloco de ações se a condição não for satisfeita. Essa estrutura nos permite decidir o que será feito se uma condição inicial não for atendida.

```
if (<condição A>)
    <comando B>;
else
    <comando C>;
```

a) Estrutura.

```
if (NotaAluno >= 5)
      printf("Aluno aprovado \n");
else
      printf("Aluno reprovado \n");
```

b) Trecho de um programa em linguagem C.

Figura 5.1 – Exemplos de estrutura de controle: condicional composta

Fonte: Elaboração própria

No exemplo a) da Figura 5.1, o **<comando B>** somente será executado se a **<condição A>** for verdadeira, mas o **<comando C>** não será executado. Se o resultado da **<condição A>** for falso, o **<comando C>** será executado, e o **<comando B>** não será executado. A **<condição A>** possui um resultado lógico. Caso seja necessário executar mais de um comando em relação a uma condição, esses comandos devem ficar dentro de um bloco. Na linguagem C, esses comandos ficam entres chaves **{ }**. A Figura 5.2 apresenta um trecho de programa em que mais de um comando será executado caso a **<condição A>** seja verdadeira ou falsa.

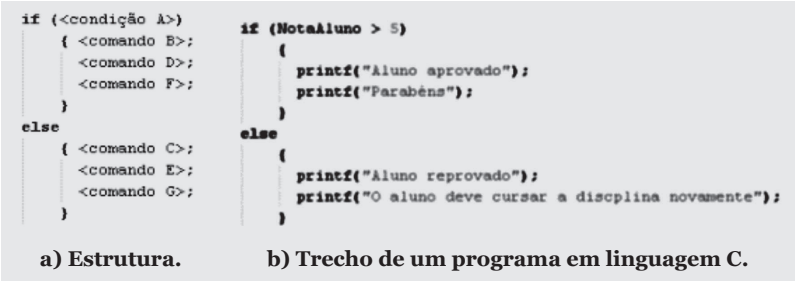


Figura 5.2 – Estrutura de controle: condicional composta

Fonte: Elaboração própria

Experimento 5

Objetivo

O objetivo desse experimento é fazer você aplicar os conhecimentos teóricos sobre a seleção composta das estruturas de controle codificados em programas na linguagem de programação C.

Orientações

Nesse experimento, você deve fazer o programa *EstruCondComp1.cpp* ler a nota do aluno e informar se ele foi aprovado ou reprovado. A seleção composta é utilizada para verificar se a nota é maior do que cinco ou não. Caso o resultado seja afirmativo, aparecerá uma mensagem dizendo que o aluno foi aprovado; se não, uma mensagem dizendo que o aluno foi reprovado.

Procedimentos

Para executar o experimento, siga este passo a passo:

1. Em um sistema operacional, execute um IDE.
2. Digite o código do programa *EstruCondComp1.cpp* (Figura 5.3).
3. Acrescente o seu nome como o aluno do programa, na linha 12.
4. Verifique se o código foi digitado corretamente.
5. Compile o programa *EstruCondComp1.cpp*.
6. Caso o programa apresente alguma mensagem de erro, verifique em qual linha, qual erro e corrija-o.
7. Caso o programa não apresente mensagem de erro, execute-o.
8. Compare a sequência dos comandos apresentados na tela com a dos comandos do programa.

***** **Início do Código do programa: *EstruCondComp1.cpp*** *****

```
1   #include <stdio.h> // Biblioteca de entrada e saida
2   #include <stdlib.h> //Biblioteca para uso do comando system("pause")
3
4   int main()
5   {
6       // Declaração das variáveis
7       int NotaAluno;
8
9       // Cabeçalho
10      printf("Universidade de Brasilia \n");
11      printf("Disciplina: 113913 -  ICC \n");
12      printf("Aluno(a): Meu Nome Completo \n\n");
13
```

a) Inclusão das bibliotecas, declaração da variável e cabeçalho do programa.

```
14      // Escrevendo mensagem na tela
15      printf("\nInforme a nota do aluno: ");
16      //Solicitação ao usuário da entrada da nota do aluno.
17      scanf("%d", &NotaAluno);
```

b) Mensagens e leitura de dados.

```
18
19      //Teste para verificar se o aluno foi aprovado.
20      if(NotaAluno >= 5){
21      //Escrevendo o resultado na tela.
22          printf("\nAluno aprovado.\nParab%cns!!!\n\n", 130);
23      }
24      else{   //Composição da estrutura condicional (caso contrário).
25      //Escrevendo o resultado na tela.
26          printf("\nAluno reprovado!!!\n\n");
27      }
28
29      // Para o sistema visualizar o resultado
30      system("PAUSE");
31      return 0;
32  }
```

c) Estrutura Condicional Composta.

```
Console
gcc -Wall "C:\ICC\EstruCondComp1.cpp" -o "C:\ICC\EstruCondComp1.exe"
gcc -Wall "C:\ICC\EstruCondComp1.cpp" -o "C:\ICC\EstruCondComp1.exe"
Process started >>>
<<< Process finished.
================ READY ================
```

d) Compilação do programa.

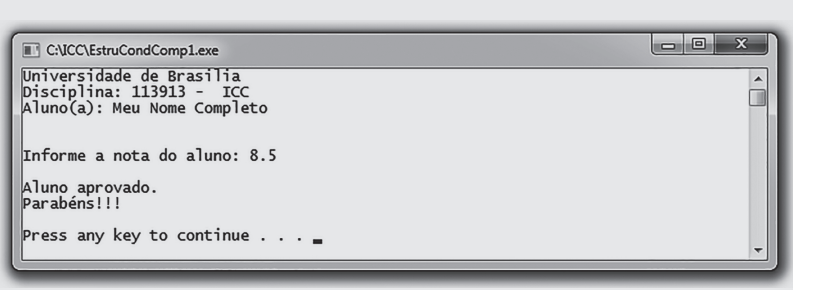
******* Fim código original *******

Figura 5.3 – Código do programa em C: *EstruCondComp1.cpp*

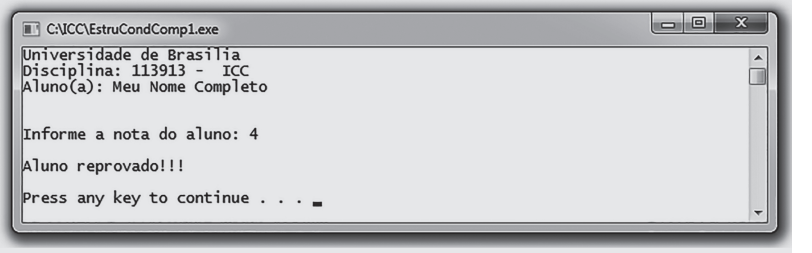
Fonte: Elaboração própria

Análise

Apresentamos o resultado da execução do programa *EstruCondSimples1.cpp* na Figura 5.4. Observe que o programa mostra um cabeçalho e, depois, solicita a nota do aluno. Em seguida, caso o aluno tenha a nota para ser aprovado, o programa escreve essa mensagem; se não, o programa informa que o aluno foi reprovado.



a) Condição com o resultado verdadeiro.



b) Condição com o resultado falso.

Figura 5.4 – Resultado do código do programa em C: *EstruCondComp1.cpp*

Fonte: Elaboração própria

Atividades de Fixação

Realize as atividades, a seguir, utilizando a estrutura condicional composta.

1. Escreva um programa que leia o número de alunos e de alunas de uma sala. Como saída, o programa deve apresentar primeiro quem estiver em maior quantidade. Por exemplo: se na sala houver mais alunos, apresente primeiro o número de alunos; se não, apresente o número de alunas e depois o de alunos.

2. Desenvolva um programa para comparar a idade de Pedro e Joana e informar quem é o mais velho. Dados de entrada: idade do Pedro e da Joana (tipo das variáveis: inteiro, valor em anos). Observação: essas pessoas possuem idades diferentes.
3. Desenvolva um programa para calcular e comparar a área de dois retângulos A e B; ele deverá dizer qual retângulo possui a maior área ou se eles possuem tamanhos iguais. Dados de entrada: tamanho da base e da altura (tipo das variáveis: inteiro, valor em centímetros). Para aprimorar seus conhecimentos, calcule o semiperímetro da área de um triângulo e o perímetro de um retângulo.

⊗ *Semi – Perímetro do triângulo* (p_{tr}) *e Perímetro do retângulo* (P_r):

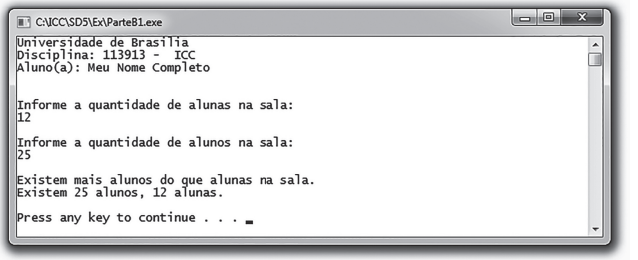
$$p_{tr} = \frac{a+b+c}{2} \quad , \quad P_r = 2h + 2b$$

⊗ *Area do triângulo e retângulos*:

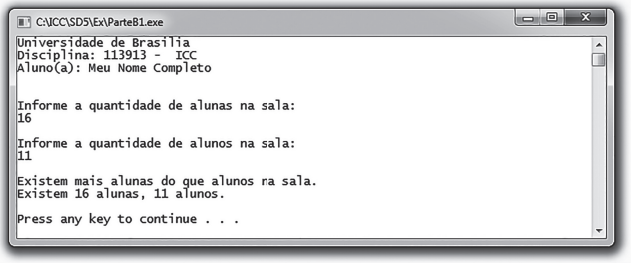
$$A_{tr} = \sqrt{P(P-a)\times(P-b)\times(P-c)} \quad , \quad A_r = b \times h$$

Respostas das Atividades de Fixação

1)

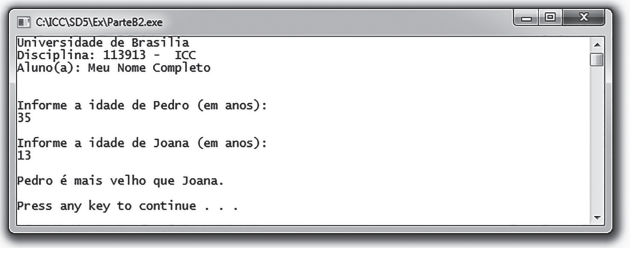


a) Condição em que há mais alunos do que alunas.



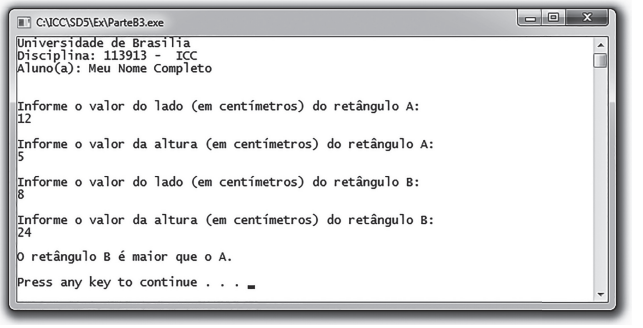
b) Condição em que há mais alunas do que alunos.

2)



a) Condição em que Pedro é mais velho do que Joana

3)



a) Condição em que o retângulo B é maior do que A.

Capítulo 6

Estrutura de Repetição Contada

Estrutura de Repetição Contada

Uma estrutura de repetição permite ao programador especificar que uma ação será repetida enquanto determinada condição lógica permanecer verdadeira. A estrutura de repetição também é um meio de assegurar que um conjunto de comandos seja repetido inúmeras vezes. Observe a Figura 6.1.

```
for (<inicialização> ; <condição>; <atualização>)
    {
        Comando1;
        Comando2;
        . . .
        ComandoN;
    }
```

a) Estrutura.

```
for (n1=1; n1<=3; n1++)
    {
        fatoirial1 = 1;
        printf("\n\nDigite um numero para o calculo do fatorial:");
        . . .
        printf("\Fatorial somente para numero positivos. \n");;
    }
```

b) Trecho de um programa em linguagem C.

Figura 6.1 – Exemplos de estrutura de repetição contada com N comandos

Fonte: Elaboração própria

No exemplo a) da Figura 6.1, temos uma estrutura de repetição contada. Essa estrutura é composta de um comando “for” que possui três parâmetros. O primeiro parâmetro é a <inicialização>, em que uma variável de controle é inicializada; o segundo é a <condição>, que, enquanto for verdadeira, um bloco de comandos será executado (no exemplo são os Comando1, Comando2... ComandoN). Observe que, caso exista

somente um comando, não há necessidade de início e fim do bloco, conforme a Figura 6.2. O terceiro parâmetro é a <atualização>, que costumamos utilizar para atualizar a variável de controle inicializada no parâmetro <inicialização>.

```
for (<inicialização> ; <condição>; <atualização>)
    Comando;
```

a) Estrutura.

```
for (n1=1; n1<=3; n1++)
    fatorial = fatorial * cont;
```

b) Trecho de um programa em linguagem C.

Figura 6.2 – Exemplos de estrutura de repetição contada com um comando

Fonte: Elaboração própria

Na Figura 6.2, apresentamos um exemplo da estrutura de repetição contada com apenas um comando, não havendo necessidade de início e fim do bloco de comandos.

Experimento 6

Objetivo

O objetivo desse experimento é apresentar a você a estrutura de repetição contada com a utilização da linguagem de programação C.

Orientações

Nesse experimento, você deve fazer o programa utilizar a estrutura de repetição contada para implementar, em linguagem C, o algoritmo que calcula a tabuada de um número. Esse programa gera o arquivo *Tabuada.cpp*.

Procedimentos

Para executar o experimento, realize este passo a passo respeitando a sequência dada nas partes A e B:

Parte A

1. Digite o código *Tabuada.cpp* (Figura 6.3).
2. Verifique se o código foi digitado corretamente.
3. Compile o programa *Tabuada.cpp*.
4. Caso apresente alguma mensagem de erro, verifique em qual linha, qual erro e corrija-o.
5. Caso o programa não apresente mensagem de erro, execute-o e obtenha o resultado parecido com o da Figura 6.4.

Parte B

1. Faça um programa para calcular o fatorial de um número.
2. Termine o código do programa *Fatorial.cpp* como iniciado na Figura 6.5 (Parte A) e use o exemplo descrito anteriormente no item 1.
3. Acrescente o seu nome como o aluno do programa na linha 13.
4. Verifique se o código foi digitado corretamente.
5. Compile o programa *Fatorial.cpp*.

6. Caso o programa apresente alguma mensagem de erro, verifique em qual linha, qual erro e corrija-o.
7. Caso o programa não apresente mensagem de erro, execute-o.
8. Compare a sequência dos comandos apresentados na tela com a sequência dos comandos do programa.
9. Salve o programa com outro nome e altere-o para que ele solicite seis vezes o número de valores a calcular o fatorial.
10. Altere o novo programa para que ele escreva o resultado do fatorial com duas casas decimais.

- É feita a declaração de variáveis em que *'n'* e *'n1'* tratam da entrada de um determinado número para o cálculo do fatorial do mesmo número e a verificação da contagem inicial e final do contador respectivamente.
- O *'cont'* é declarado para contagem dos valores do cálculo do fatorial do número *'n'*.

******* Início do Código do programa: *Tabuada.cpp* *******

```
1   #include <stdio.h> // Biblioteca de entrada e saida
2   #include <stdlib.h> //Biblioteca para uso do comando system("pause")
3
4   int main()
5   {
6       // Declaração das variáveis
7       int n, cont, tabuada;
8
9       // Cabeçalho
10      printf("Universidade de Brasilia \n");
11      printf("Disciplina: 113913 -  ICC \n");
12      printf("Aluno(a): Meu Nome Completo \n\n");
13
```

a) Inclusão das bibliotecas, declaração da variável e cabeçalho do programa.

```
14      n = 6;  // Atribuindo à variável n o valor 6.
15
16      // Escrevendo mensagem na tela
17      printf("\n\nA tabuada do n%cmero %d %c:\n\n\n", 163, n, 130);
18
```

b) Mensagens e leitura de dados.

```
19      // Loop para o cálculo da tabuada.
20      for(cont = 0; (cont <= 10); cont++){
21          tabuada = cont * n;
22          //Impressão dos resultados.
23          printf("\t%d * %d = %d\n", n, cont, tabuada);
24      }
25      // Pulando uma linha
26      printf("\n");
27
28      // Para o sistema visualizar o resultado
29      system("PAUSE");
30      return 0;
31  }
```

c) Estrutura de Repetição Contada.

```
Console
gcc -Wall "C:\ICC\tabuada.cpp" -o "C:\ICC\tabuada.exe"
gcc -Wall "C:\ICC\tabuada.cpp" -o "C:\ICC\tabuada.exe"
Process started >>>
<<< Process finished.
================ READY ================
```

d) Compilação do programa.

******* Fim código original *******

Figura 6.3 – Código do programa em C: *Tabuada.cpp*

Fonte: Elaboração própria

Análise

Na Figura 6.4, mostramos o resultado da execução do programa *Tabuada.cpp* (Parte A). Observe que o programa apresenta um cabeçalho, em seguida mostra o número 6, que foi utilizado para o cálculo da tabuada de multiplicação desse número.

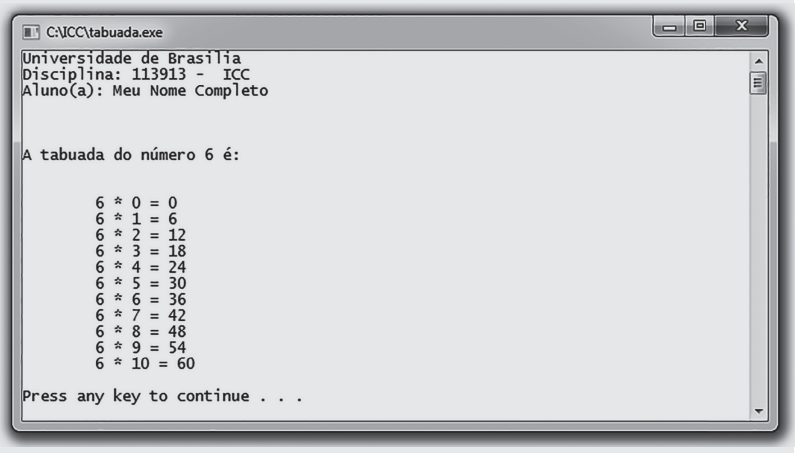


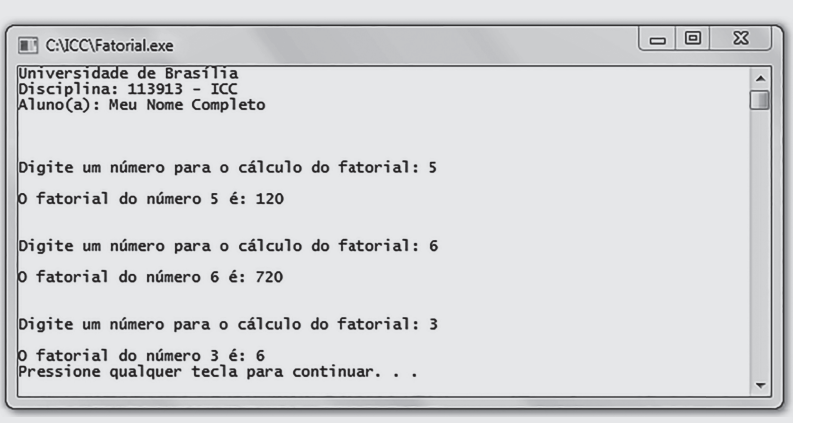
Figura 6.4 – Exemplo de resultado para tabuada de vezes do número 6

Fonte: Elaboração própria

Na Figura 6.5, temos o resultado esperado do fatorial de três números aleatórios (Parte B).

```
6       // Declaração das variáveis
7       int n, n1, cont;
8       double fatorial;
9
10      // Cabeçalho
11      printf("Universidade de Brasilia \n");
12      printf("Disciplina: 113913 -  ICC \n");
13      printf("Aluno(a): Meu Nome Completo \n\n");
14
15      // Loop para o cálculo do fatorial de 3 fatorias.
16      for(n1=1; (n1<=3); n1++){
17          // Inicialização da variável
18          fatorial = 1;
19          //Entrada de dados.
20          printf("\n\nDigite um n%cmero para o c%clculo do fatorial:\n", 163, 160);
21          scanf("%d", &n);
22
```

a) Trecho do programa *Fatorial.cpp*.



b) Resultado do programa.

Figura 6.5 – Exemplo parcial de código e resultado esperado do fatorial de três números aleatórios

Fonte: Elaboração própria

Atividades de Fixação

1. Dado um número, crie um programa C que escreva todos os números ímpares menores e/ou iguais a esse número e maiores ou igual a um. Assuma que o número informado é positivo.
2. Dado um conjunto de N números, crie um programa C que calcule e mostre sua média aritmética.
3. A conversão de graus Fahrenheit para Celsius é obtida por:

   Tc={(Tf-32)*(5/9)}, em que Tc é a temperatura em graus Celsius e Tf em Fahrenheit. Faça um programa C que calcule e escreva um programa que imprima uma tabela de graus Fahrenheit e graus Celsius, cujos graus variem de 50 a 65, de 1 em 1.

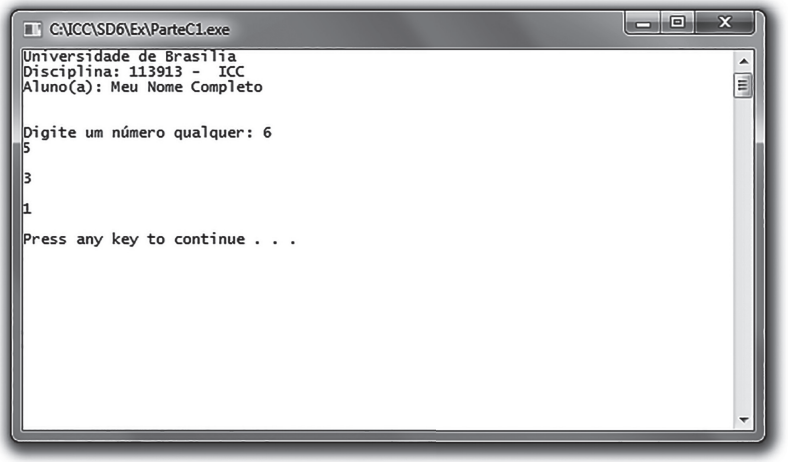
4. Faça um programa C que leia dez números que representem as notas de dez alunos e obtenha:

a) a soma dos números;
b) a média dos números;
c) o maior número;
d) o menor número.

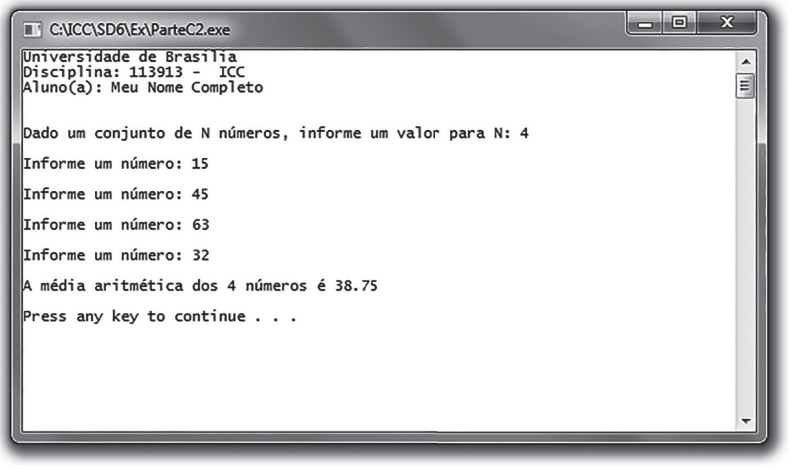
Assuma que as notas são informadas corretamente no intervalo de 1 a 10.

Respostas das Atividades de Fixação

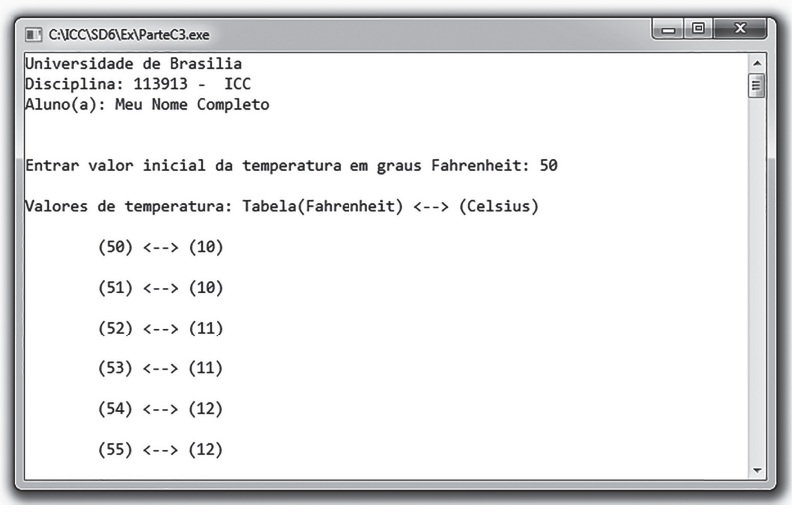
1)

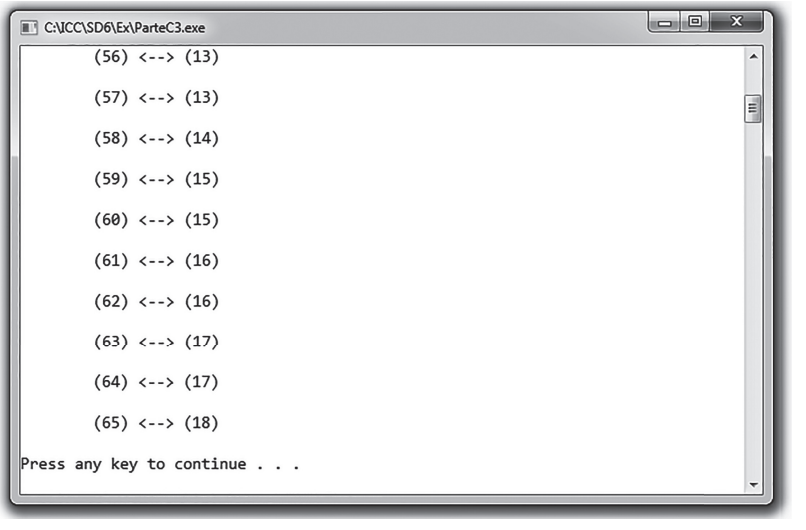


2)

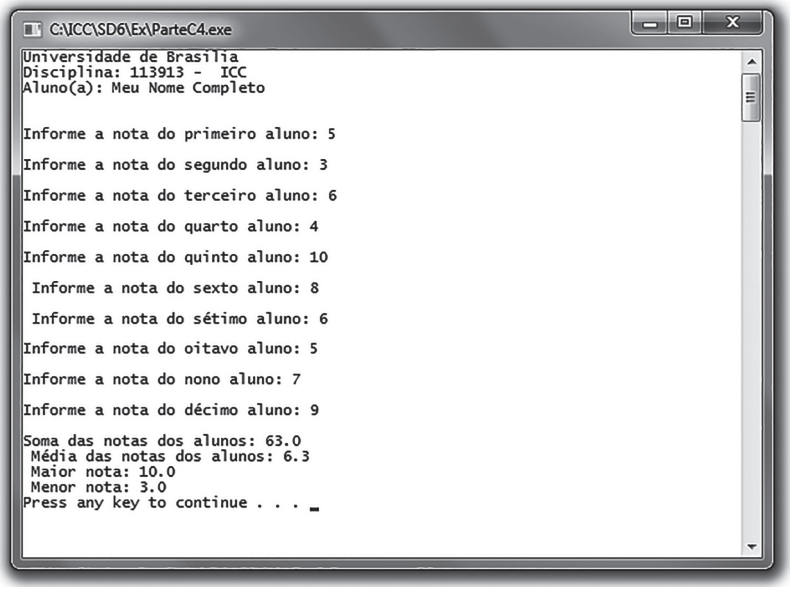


3)


C:\ICC\SD6\Ex\ParteC3.exe
Universidade de Brasilia
Disciplina: 113913 - ICC
Aluno(a): Meu Nome Completo
Entrar valor inicial da temperatura em graus Fahrenheit: 50
Valores de temperatura: Tabela(Fahrenheit) <--> (Celsius)
(50) <--> (10)
(51) <--> (10)
(52) <--> (11)
(53) <--> (11)
(54) <--> (12)
(55) <--> (12)


C:\ICC\SD6\Ex\ParteC3.exe
(56) <--> (13)
(57) <--> (13)
(58) <--> (14)
(59) <--> (15)
(60) <--> (15)
(61) <--> (16)
(62) <--> (16)
(63) <--> (17)
(64) <--> (17)
(65) <--> (18)
Press any key to continue . . .

4)


C:\ICC\SD6\Ex\ParteC4.exe
Universidade de Brasilia
Disciplina: 113913 - ICC
Aluno(a): Meu Nome Completo
Informe a nota do primeiro aluno: 5
Informe a nota do segundo aluno: 3
Informe a nota do terceiro aluno: 6
Informe a nota do quarto aluno: 4
Informe a nota do quinto aluno: 10
Informe a nota do sexto aluno: 8
Informe a nota do sétimo aluno: 6
Informe a nota do oitavo aluno: 5
Informe a nota do nono aluno: 7
Informe a nota do décimo aluno: 9
Soma das notas dos alunos: 63.0
Média das notas dos alunos: 6.3
Maior nota: 10.0
Menor nota: 3.0
Press any key to continue . . .

Capítulo 7

Estrutura de Repetição Condicional com Teste no Início

Condicional com Teste no Início

Uma estrutura de repetição permite ao programador especificar que uma ação será repetida enquanto determinada condição lógica permanecer verdadeira. A estrutura de repetição assegura que um conjunto de um ou mais comandos seja repetido inúmeras vezes.

```
while (<condição>)
{
    Comando1;
    Comando2;
    Comando3;
    ...
    ComandoN;
}
```

a) Estrutura.

```
while((sexo != 'm')){
    printf( "\nErro! ");
    printf("\nInforme o sexo novamente: ");
    scanf("%c",&sexo);
}
```

b) Trecho de um programa em linguagem C.

Figura 7.1 – Exemplos de estrutura de controle: condicional com teste no início

Fonte: Elaboração própria

No exemplo (a) da Figura 7.1, os comandos **<Comando1>**, **<Comando2>**, **<Comando3>**;**...** e **<ComandoN>** somente serão executados enquanto a **<condição>** for verdadeira. Ou seja, nessa estrutura, há uma condição logo no início que, enquanto for satisfeita (verdadeira), o conjunto de ações limitadas pelo bloco da repetição será executado. Quando a condição for falsa, o comando será abandonado, prosseguindo para o próximo comando na sequência. Observe que, caso exista somente um comando, não há necessidade de chaves delimitando o bloco de comandos **do while**.

Experimento 7

Objetivo

O objetivo desse experimento é fazer você compreender a estrutura de repetição condicional com teste no início codificada na linguagem de programação C.

Orientações

Nesse experimento, você deve fazer o programa *CriticaEntrada.cpp* utilizar a estrutura de repetição com teste no início, em linguagem C, como mostramos na Figura 7.2. Esse programa deve ler duas informações de entrada: o sexo e o ano. O enquanto (while) deve ser utilizado para assegurar que os valores de sexo lidos sejam "m", "M", "f" ou "F". Para ler um caractere, o programa deve utilizar **%1s** (sendo "1" o tamanho e "s" de string, que significa palavra em português).

Procedimentos

Para executar o experimento, realize este passo a passo respeitando a sequência dada nas partes A e B a seguir:

1. Em um sistema operacional, execute um IDE.
2. Digite o código do programa *CriticaEntrada.cpp* (Figura 7.2).
3. Acrescente ao código o cabeçalho contendo a Universidade, o curso e o seu nome como o aluno que escreveu o programa.
4. Verifique se o código foi digitado corretamente.
5. Compile o programa *CriticaEntrada.cpp*.

6. Caso o programa apresente alguma mensagem de erro, verifique em qual linha, qual erro e corrija-o.
7. Caso o programa não apresente mensagem de erro, execute-o.
8. Compare a sequência dos comandos apresentados na tela com a sequência dos comandos do programa.
9. Salve o programa com outro nome.
10. Execute o programa.

******* Início do Código do programa: *CriticaEntrada.cpp* *******

```
1  #include <stdio.h> // Biblioteca de entrada e saida
2  #include <stdlib.h> //Biblioteca para uso do comando system("pause")
3
4  int main(){
5
6      //Declaração das variáveis
7      int ano;
8      char sexo;
9
10     // Cabeçalho
11     printf("Universidade de Brasilia \n");
12     printf("Disciplina: 113913 -  ICC \n");
13     printf("Aluno(a): Meu Nome Completo \n\n");
14
```

a) Inclusão das bibliotecas, declaração das variáveis e cabeçalho do programa.

```
15     // Solicitação ao usuário da entrada do sexo.
16     printf("\nInforme o sexo:\n");
17     // Leitura de dados.
18     scanf("%1s", &sexo);
19
```

b) Mensagens e leitura de dados.

```
20
21     while((sexo!='f') && (sexo!='F') && (sexo!='m') && (sexo!='M')){
22         // Solicitação ao usuário da entrada do sexo.
23         printf("\nErro!!!\nInforme o sexo, novamente:\n");
24         // Leitura de dados.
25         scanf("%1s", &sexo);
26     }
27
```

c) Estrutura de Repetição: Condicional com teste no início.

```
28      // Solicitação ao usuário da entrada do sexo.
29      printf("\nInforme o ano de nascimento do usu%crio:\n", 160);
30      // Leitura de dados.
31      scanf("%d", &ano);
32
33      // Impressão dos resultados.
34      printf("\n%d - %c\n", ano, sexo);
35
36      // Para o sistema visualizar o resultado
37      system("PAUSE");
38      return 0;
39  }
```

d) Apresentação e formatação dos resultados na tela.

```
Console
gcc -Wall "C:\ICC\CriticaEntrada.cpp" -o "C:\ICC\CriticaEntrada.exe"
gcc -Wall "C:\ICC\CriticaEntrada.cpp" -o "C:\ICC\CriticaEntrada.exe"
Process started >>>
<<< Process finished.
================ READY ================
```

e) Compilação do programa.

******* Fim código original *******

Figura 7.2 – Código do programa em C: *CriticaEntrada.cpp*

Fonte: Elaboração própria

Análise

Apresentamos o resultado da execução do programa *CriticaEntrada.cpp* na Figura 7.3. Observe que o programa mostra um cabeçalho e, em seguida, solicita o sexo do usuário e a data de nascimento. Por fim, o programa informa o resultado dos dados entrados anteriormente pelo usuário.

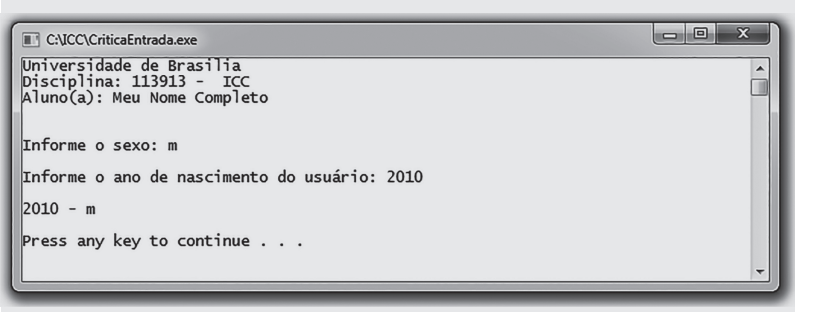


Figura 7.3 – Resultado do código do programa em C: *CriticaEntrada.cpp*

Fonte: Elaboração própria

Atividades de Fixação

1. Dado um número, crie um programa C que escreva todos os números ímpares menores que esse número e maiores do que um. Assuma que o número informado é positivo.
2. Faça um programa que exiba a tabuada dos números de 10 a 20. Por exemplo:

   10x0=0

   10x1=10

   ....

   10x10=100

   até chegar à tabuada de 20,

   20x0=0

   20x1=20

   ....

   20x10=200

3. Um funcionário de uma empresa recebe aumento salarial anualmente. Sabemos que:

   a) esse funcionário foi contratado em 1995, com salário inicial de R$ 1.000,00;

   b) em 1996, recebeu aumento de 1,5% sobre seu salário inicial;

   c) a partir de 1997 (inclusive), os aumentos salariais sempre corresponderam ao dobro do percentual do ano anterior.

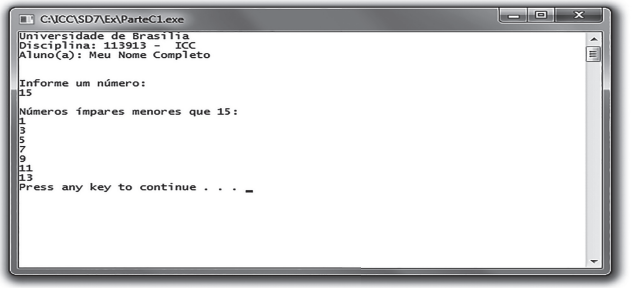
   Faça um programa que determine o salário desse funcionário até o ano 2000 (se o aluno tiver uma excelente máquina, poderá realizar a operação até o ano final desejado).

4. Faça um programa que leia dez conjuntos de dois valores: o primeiro representando a matrícula do aluno, e o segundo representando a sua altura em centímetros. Encontre o aluno mais alto e o mais baixo. Mostre o número do aluno mais alto e o número do aluno mais baixo, cada um com sua altura.

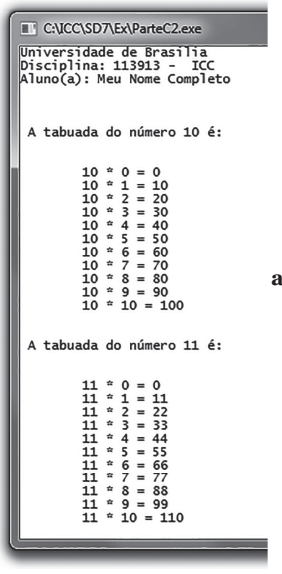
5. Faça um programa que mostre todos os números pares existentes entre 1 e 48.

Respostas das Atividades de Fixação

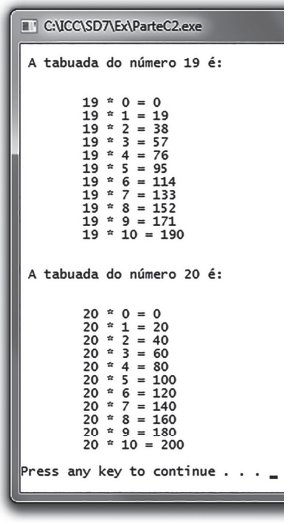
1)



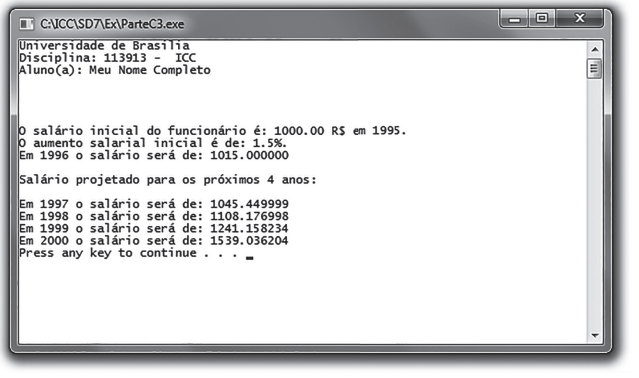
2)



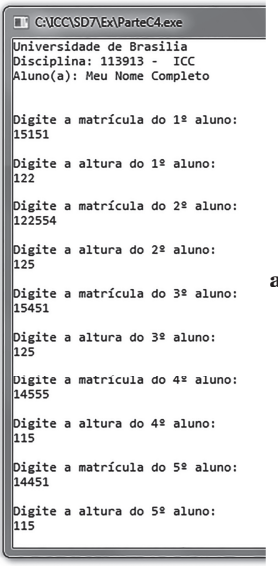
até



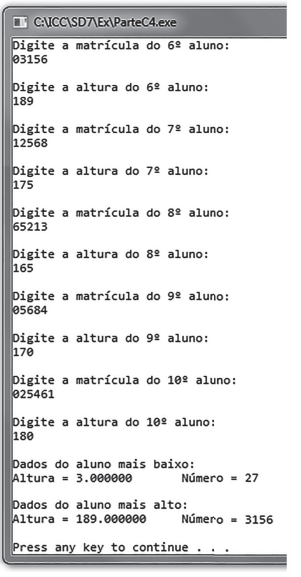
3)



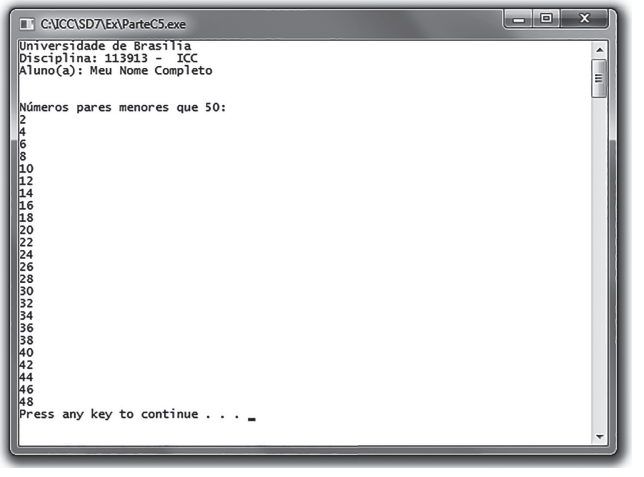
4)


C:\ICC\SD7\Ex\ParteC4.exe
Universidade de Brasilia
Disciplina: 113913 - ICC
Aluno(a): Meu Nome Completo
Digite a matrícula do 1º aluno:
15151
Digite a altura do 1º aluno:
122
Digite a matrícula do 2º aluno:
122554
Digite a altura do 2º aluno:
125
Digite a matrícula do 3º aluno:
15451
Digite a altura do 3º aluno:
125
Digite a matricula do 4º aluno:
14555
Digite a altura do 4º aluno:
115
Digite a matrícula do 5º aluno:
14451
Digite a altura do 5º aluno:
115

até


C:\ICC\SD7\Ex\ParteC4.exe
Digite a matrícula do 6º aluno:
03156
Digite a altura do 6º aluno:
189
Digite a matrícula do 7º aluno:
12568
Digite a altura do 7º aluno:
175
Digite a matrícula do 8º aluno:
65213
Digite a altura do 8º aluno:
165
Digite a matrícula do 9º aluno:
05684
Digite a altura do 9º aluno:
170
Digite a matrícula do 10º aluno:
025461
Digite a altura do 10º aluno:
180
Dados do aluno mais baixo:
Altura = 3.000000 Número = 27
Dados do aluno mais alto:
Altura = 189.000000 Número = 3156
Press any key to continue . . .

5)


C:\ICC\SD7\Ex\ParteC5.exe
Universidade de Brasilia
Disciplina: 113913 - ICC
Aluno(a): Meu Nome Completo
Números pares menores que 50:
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
Press any key to continue . . .

Capítulo 8

Estrutura de Repetição Condicional com Teste no Final

Condicional com Teste no Final

Uma estrutura de repetição permite ao programador especificar que uma ação será repetida enquanto determinada condição lógica permanecer verdadeira. Logo, a estrutura de repetição assegura que um conjunto de um ou mais comandos seja repetido inúmeras vezes.

```
do{
    Comando1;
    Comando2;
    Comando3;
    ...
    ComandoN;
}while (<condição>)
```

a) Estrutura.

```
do {
    printf("\ndigite um elemento da seqüência:");
    scanf("%d", &x);
    soma = soma + x;
} while (x != 0);
```

b) Trecho de um programa em linguagem C.

Figura 8.1 – Exemplos de estrutura de controle: condicional com teste no final

Fonte: Elaboração própria

No exemplo (a) da Figura 8.1, os comandos de **<Comando1>** até **<ComandoN>** serão executados enquanto a **<condição>** for verdadeira. Ou seja, nessa estrutura, há uma condição logo no final de um bloco de comandos que, enquanto for satisfeita (verdadeira), o conjunto de ações limitadas pelo bloco da repetição será executado. Quando a condição for

falsa, o comando será abandonado seguindo para o próximo comando na sequência de comandos do programa. Observamos que, caso exista somente um comando, não há a necessidade das chaves delimitando o bloco de comandos **do while**.

Experimento 8

Objetivo

O objetivo desse experimento é apresentar a você a estrutura de repetição condicional com teste no final codificada na linguagem de programação C.

Orientações

Nesse experimento, você deve fazer o programa *SomaElementos.cpp* utilizar a estrutura de repetição com teste no final, em linguagem C, como mostramos na Figura 8.2. Nesse programa, deve usar o comando **do while** para assegurar que sejam somados os valores recebidos na entrada do programa enquanto ele não for igual a 0. Caso o valor entrado seja igual a 0, o programa *SomaElementos.cpp* deve finalizar a soma dos elementos digitados pelo usuário.

Procedimentos

Para executar esse experimento, realize este passo a passo respeitando a sequência dada nas partes A e B:

Parte A

1. Em um sistema operacional, execute um IDE.
2. Digite o código do programa *SomaElementos.cpp* (Figura 8.2).

3. Acrescente ao código o cabeçalho contendo a Universidade, o curso e o seu nome como o aluno que escreveu o programa.
4. Verifique se o código foi digitado corretamente.
5. Compile o programa *SomaElementos.cpp*.
6. Caso o programa apresente alguma mensagem de erro, verifique em qual linha, qual erro e corrija-o.
7. Caso o programa não apresente mensagem de erro, execute-o.
8. Compare a sequência dos comandos apresentados na tela com a sequência dos comandos do programa.
9. Salve o programa com outro nome.
10. Altere o novo programa para implementar a multiplicação dos elementos e o término da multiplicação ser dado pela entrada do dígito 1.

Parte B

1. Faça um programa que calcule o valor da soma: S = 1/1+3/2+5/3+7/4+ .... + 99/50.
2. Verifique se o código foi digitado corretamente.
3. Compile o programa com o nome de *SomadeFracoes.cpp*.
4. Caso apresente alguma mensagem de erro, verifique em qual linha, qual erro e corrija-o.
5. Caso o programa não apresente mensagem de erro, execute-o.

******* Início do Código do programa: *SomaElementos.cpp* *******

```
1  #include <stdio.h> // Biblioteca de entrada e saida
2  #include <stdlib.h> //Biblioteca para uso do comando system("pause")
3
4  int main(){
5
6      //Declaração das variáveis
7      int soma = 0, i;
8
9      // Cabeçalho
10     printf("Universidade de Brasilia \n");
11     printf("Disciplina: 113913 -  ICC \n");
12     printf("Aluno(a): Meu Nome Completo \n\n");
13
```

a) Inclusão das bibliotecas, declaração das variáveis e cabeçalho do programa.

```
14     // Estrutura que permite a repetição da leitura e soma dos elementos
15     // até que se encontre o elemento final da sequência.
16     do{
17         // Solicitação ao usuário da entrada de um novo elemento da sequência.
18         printf("\nDigite um elemento da sequ%cncia, \"ou digite 0 para terminar\": ", 136);
19         // Leitura de dados.
20         scanf("%d", &i);
21         // Soma dos elementos
22         soma += i;
23     }while(i != 0);
24
```

b) Estrutura de Repetição: Condicional com teste no final.

```
25         //Impressão dos resultados.
26         printf("\nO somat%crio dos elementos %c: %d. ", 160, 130, soma);
27         printf("\n\n");
28
29         // Para o sistema visualizar o resultado
30         system("PAUSE");
31         return 0;
32 }
```

c) Mensagens e leitura de dados.

```
Console
gcc -Wall "C:\ICC\SomaElementos.cpp" -o "C:\ICC\SomaElementos.exe"
gcc -Wall "C:\ICC\SomaElementos.cpp" -o "C:\ICC\SomaElementos.exe"
Process started >>>
<<< Process finished.
================ READY ================
```

d) Compilação do programa.

******* Fim código original *******

Figura 8.2 – Código do programa em C: *SomaElementos.cpp*

Fonte: Elaboração própria

Análise

Apresentamos o resultado da execução do programa *SomaElementos.cpp* na Figura 8.3. Observe que o programa mostra um cabeçalho e, em seguida, solicita um número de sequência a ser informado pelo usuário. Por fim, o programa informa o resultado dos dados digitados pelo usuário.

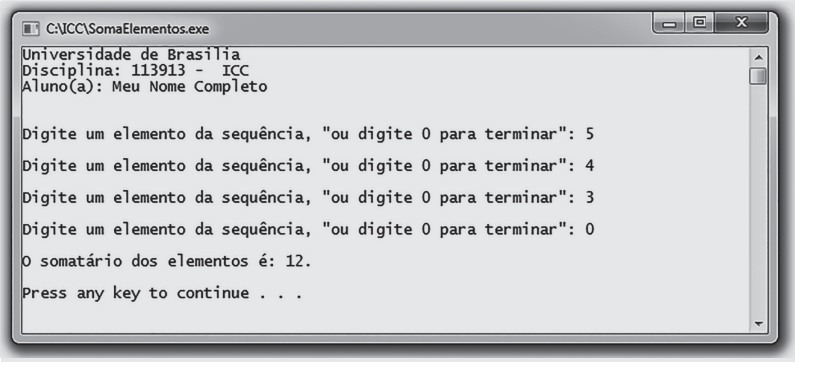


Figura 8.3 – Resultado do código do programa em C: *SomaElementos.cpp*
Fonte: Elaboração própria

Apresentamos, nas Figura 8.4 e 8.5, os resultados da Parte B do procedimento de execução do programa *SomaElementos.cpp*.

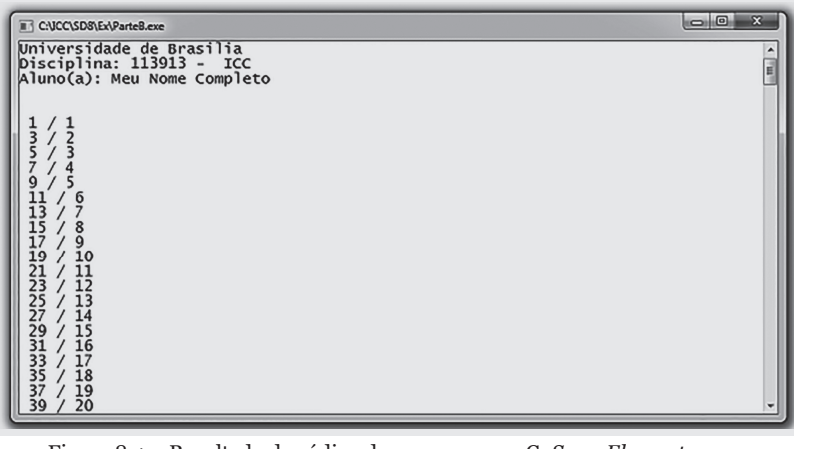


Figura 8.4 – Resultado do código do programa em C: *SomaElementos.cpp*
Fonte: Elaboração própria

Na sequência, temos os valores 41/21... 57/29... até chegar à soma pretendida, que é 50, como mostramos na Figura 8.5.

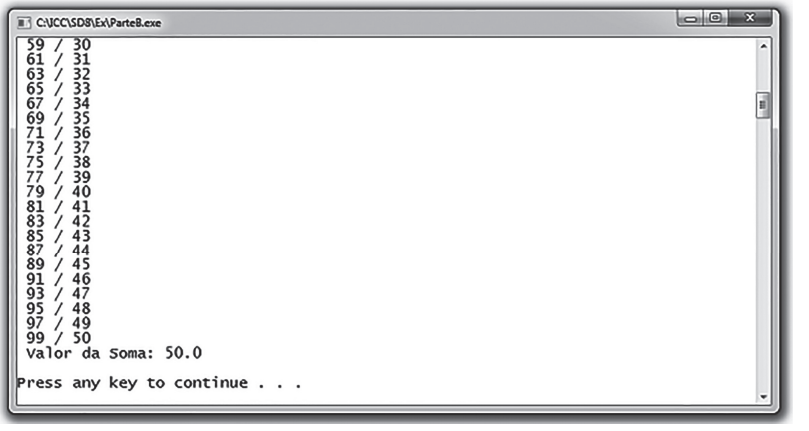


Figura 8.5 – Resultado final do código do programa em C: *SomadeFracoes.cpp*
Fonte: Elaboração própria

Atividades de Fixação

Realize as atividades, a seguir, utilizando a estrutura de repetição com teste no final.

1. Dado um número, crie um programa C que escreva todos os números ímpares menores que esse número e maiores do que um. Assuma que o número informado é positivo.
2. Dado um conjunto de N números, faça um programa C que calcule e mostre a sua média aritmética.
3. Faça um programa C que leia dez números que representem as notas de dez alunos de uma disciplina. As notas variam de zero até dez (0 a 10). O programa deve validar a entrada de dados e obter:

   a) a soma das notas;

   b) a média das notas;

   c) a maior nota;

   d) a menor nota.

   Assuma que as notas são informadas corretamente no intervalo de 1 a 10.

4. Faça um programa que leia um conjunto que contém dois valores: o primeiro representando a matrícula do aluno, e o segundo representando a sua altura em centímetros. Encontre o aluno mais alto e o mais baixo. Mostre o número do aluno mais alto e o número do aluno mais baixo e a altura de cada um.
5. Faça um programa que mostre todos os números pares existentes entre 1 e 48.

Respostas das Atividades de Fixação

1)

A tela de resposta desta atividade é idêntica à mostrada no Capítulo 6, atividade 1.

2)

A tela de resposta desta atividade é idêntica à mostrada no Capítulo 6, atividade 2.

3)

A tela de resposta desta atividade é idêntica à mostrada no Capítulo 6, atividade 4.

4)

A tela de resposta desta atividade é idêntica à mostrada no Capítulo 7, atividade 4.

5)

A tela de resposta desta atividade é idêntica à mostrada no Capítulo 7, atividade 5.

Capítulo 9

Funções

Funções

Os programas podem se tornar grandes e complexos para resolver alguns problemas da vida real, podendo exigir códigos grandes, difíceis de ler e com grande repetição de trechos.

Em programação, é uma prática comum dividir programas grandes em programas menores e de mais fácil compreensão/manutenção, possibilitando também a reutilização de códigos. Para executar tal divisão, fazemos uso da função.

Uma função é um conjunto de instruções projetadas para realizar uma tarefa em particular. E cada função deve possuir um nome para referenciá-la.

Todo programa em C tem pelo menos uma função (**main()**), que pode chamar outras funções ou outros procedimentos, que podem ainda chamar outras ou outros e assim por diante. Por exemplo, a função **printft()** é muito utilizada por programadores que desconhecem detalhes da sua programação.

Toda função tem um nome na execução de um programa, e a execução do programa é desviada para realizar as instruções dessa função quando esse nome é encontrado. Ao terminar a execução dessa função, continua a execução do programa que originou a chamada da função.

Na linguagem C, além das funções, existem os procedimentos. Os procedimentos também são funções, mas diferem por não retornar valores, ou seja, o "**retorno**" dos procedimentos é vazio, eles são definidos como tipo **void**.

As funções podem ser declaradas antes ou depois do escopo do programa principal (**main()**). Para que as funções sejam declaradas depois do programa principal (**main()**), é necessário que haja uma referência/um protótipo antes do programa principal (**main()**).

Quando a função é declarada depois do programa principal (**main()**), o protótipo da função deve ser usado para o compilador saber que essa função existe. A declaração, denominada protótipo da função, informa ao compilador qual o seu tipo de retorno, seu nome e os seus parâmetros, enquanto sua declaração diz o que ela executa.

Também podemos passar variáveis para a função cujos valores recebidos são chamados de argumentos da função e declarar variáveis locais dentro do corpo da função. Essas variáveis locais somente existem dentro da função. Quando a função termina a sua execução, essas variáveis locais são desalocadas.

A seguir, observe a declaração de uma função.

```
tipo_do_retorno_da_função  nome_da função (argumento_da_função){
    Comandos_da_função;
    Comandos_da_função;
    return(retorno_da_função);
}

int main(){
Comandos_do_programa;
nome_da_função (argumentos_da_função);
Comandos_do_programa;
}
```

a) Exemplo de função com declaração antes do programa.

```c
// procedimento de cabeçalho sem retorno de função
void cabecalho(){
    printf("Universidade de Brasilia \n");
    printf("Disciplina: 113913 - ICC \n");
    printf("Aluno(a): Meu Nome Completo \n\n");
}

int main(){
    // Chamada e Inclusão do procedimento cabeçalho
    cabecalho();
    // Comandos do programa
    system("pause");
    return 0;
}
```

b) Trecho de um programa em linguagem C com procedimento.

Figura 9.1 – Exemplo de declaração de funções sem protótipo de função

Fonte: Elaboração própria

Na Figura 9.1, temos um exemplo de declaração de uma função antes da chamada dessa função. A Figura 9.1 b) exemplifica a declaração de uma função sem retorno, do tipo **void**, caso em que essa função é chamada de procedimento. O <**tipo_do_retorno_da_função**> descreve o tipo da função referente ao seu retorno via comando **return()**. Esse tipo pode ser **float**, **int**, **double**, entre outros.

O **<nome_da função (argumento_da_função)>** permite receber um nome genérico criado pelo usuário, por exemplo: verificaCPF(doublenumCPF). Entre os parênteses dessa função, temos **<argumento_da_função>**, que permite a função receber dados no momento da chamada da função.

Os **<comandos>** na Figura 9.1 a) correspondem às instruções **printf()** do procedimento cabeçalho, que irá escrever na tela os dados inseridos dentro da função **printf()**, conforme a Figura 9.1 b).

A Figura 9.2 mostra o exemplo anterior, mas com a declaração de uma função depois da função principal **main()**.

```
tipo_do_retorno_da_função  nome_da função (argumento_da_função);

int main(){
Comandos_do_programa;
nome_da_função (argumentos_da_função);
Comandos_do_programa;
}
tipo_do_retorno_da_função  nome_da função (argumento_da_função){
    Comandos_da_função;
    Comandos_da_função;
    return(retorno_da_função);
}
```

a)Exemplo de função com protótipo de função e a declaração da função depois do programa main().

```
// Protótipo do procedimento cabeçalho
void cabecalho();

int main(){
    // Chamada e Inclusão do procedimento cabeçalho
    cabecalho();
    // Comandos do programa
    system("pause");
    return 0;
  }
// procedimento de cabeçalho sem retorno de função
void cabecalho (){
    printf("Universidade de Brasilia \n");
    printf("Disciplina: 113913 -  ICC \n");
    printf("Aluno(a): Meu Nome Completo \n\n");
}
```

b) Trecho de um programa em linguagem C com protótipo de função.

Figura 9.2 – Exemplo de declaração de funções com protótipo de função

Fonte: Elaboração própria

Salientamos que a declaração de uma função depois da função principal **main()** exige o protótipo antes do corpo da função **main()**, conforme podemos ver na Figura 9.2 b).

Experimento 9

Objetivo

O objetivo desse experimento é fazer você praticar a programação de funções na linguagem de programação C.

Orientações

Nesse experimento, você deve fazer o programa *Func.cpp* realizar a conversão de graus Fahrenheit para Celsius via equação Tc={(Tf-32)x(5/9)}, em que Tc é a temperatura em graus Celsius e Tf, em Fahrenheit.

Procedimentos

Para executar o experimento, realize este passo a passo respeitando a sequência dada nas partes A e B:

Parte A

1. Em um sistema operacional, execute um IDE.
2. Digite o código do programa *Func.cpp* (Figura 9.3).
3. Altere no código do cabeçalho o seu nome como o aluno que escreveu o programa.
4. Verifique se o código foi digitado corretamente.
5. Compile o programa *Func.cpp*.
6. Caso o programa apresente alguma mensagem de

erro, verifique em qual linha, qual erro e corrija-o.

7. Caso o programa não apresente mensagem de erro, execute-o.
8. Compare a sequência dos comandos apresentados na tela com a sequência dos comandos do programa.
9. Salve o programa com outro nome.

Parte B

1. Escreva um programa que solicite dois números ao usuário e apresente na tela o resultado da sua soma e o dobro de cada um deles. Esse programa deve possuir duas funções: uma para calcular a soma, e outra para calcular o dobro desses números.

******* Início do Código do programa: *Func.cpp* *******

```
1  #include <stdio.h> // Biblioteca de entrada e saida
2  #include <stdlib.h> //Biblioteca para uso do comando system("pause")
3
4  // Procedimento Cabeçalho
5  void cabecalho (){
6      printf("Universidade de Brasilia \n");
7      printf("Disciplina: 113913 -  ICC \n");
8      printf("Aluno(a): Meu Nome Completo \n\n");
9  }
10
11 // Protótipo da função de conversão
12 float converter (float tempfar);
13
```

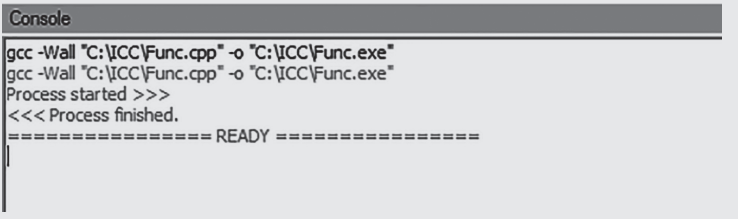
a) Inclusão das bibliotecas e declaração dos procedimentos cabeçalho e do protótipo de função.

```
14 int main(){
15
16     // Declaração das variaveis
17     float tempfar,tempcel;
18
19     // Chamada e Inclusão do procedimento cabeçalho
20     cabecalho();
21
22     // Recebe temperatura em °F
23     printf("Digite a temperatura em Fahrenheit: ");
24     scanf("%f", &tempfar);
25
26     // Converte a temperatura de °F para °C
27     tempcel = converter(tempfar);
28
29     // Exibe a temperatura em °C
30     printf("\nA temperatura Em Celsius %c %.2f \n\n", 130, tempcel);
31
32     // Para o sistema visualizar o resultado
33     system("PAUSE");
34     return 0;
35 }
```

b) Função main() invocando o procedimento cabeçalho() e função converter (argumento tipo real).

```
36
37 // Definição do corpo da função
38 float converter(float tempfar)
39 {
40     float tempcel1;
41     tempcel1 = ((tempfar - 32)*5)/9;
42     return tempcel1;
43 }
```

c) Declaração da função converter (argumento tipo real).



d) Compilação do programa *Func.cpp*.

******* Fim código original *******

Figura 9.3 – Código do Programa: *Func.cpp*

Fonte: Elaboração própria

Análise

Apresentamos o resultado da execução do programa *Func.cpp* na Figura 9.4. Observe que o programa pede ao usuário o valor da temperatura em Fahrenheit (ºF) para calcular, via chamada da função *converter(tempfar)*, o valor da temperatura em graus Celsius (ºC) e informá-la ao usuário.

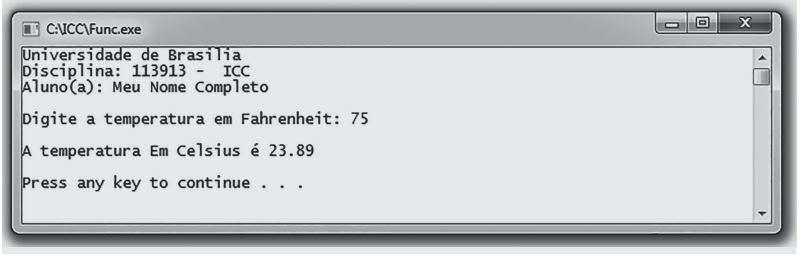


Figura 9.4 – Execução do programa: *Func.cpp*

Fonte: Elaboração própria

A Figura 9.5 apresenta o resultado da execução do programa *Func.cpp* referente à Parte B do experimento.

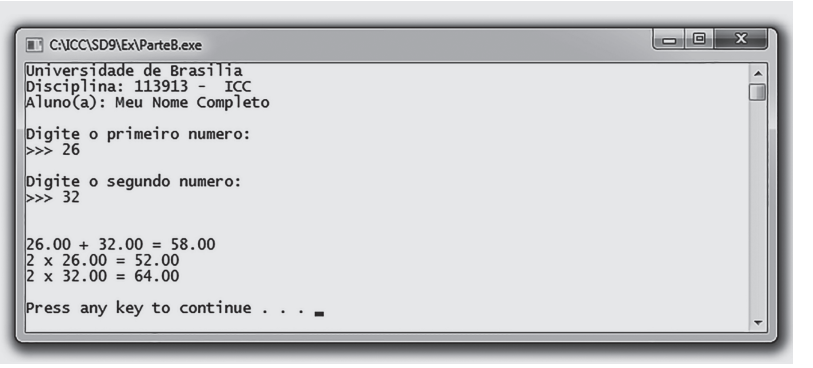


Figura 9.5 – Resultado da execução do programa da Parte B do procedimento

Fonte: Elaboração própria

Observe que o programa solicita ao usuário dois valores e, em seguida, apresenta a soma desses dois números e o seu dobro.

Atividades de Fixação

Realize as atividades, a seguir, utilizando o conteúdo sobre funções.

1. Escreva um programa que solicite dois números do tipo inteiro distintos do usuário e apresente na tela o maior deles. Esse programa deve possuir uma função para verificar qual é o maior número.
2. Desenvolva um programa para calcular e comparar a área de dois retângulos A e B. O programa deverá dizer qual retângulo possui a maior área ou se eles são de tamanhos iguais. Esse programa deve possuir uma função para calcular a área do retângulo. Dados de entrada: tamanho da base e da altura (tipo das variáveis: inteiro, valor em centímetros).
3. Escreva um programa que solicite a temperatura em Celsius ao usuário e apresente na tela o resultado da

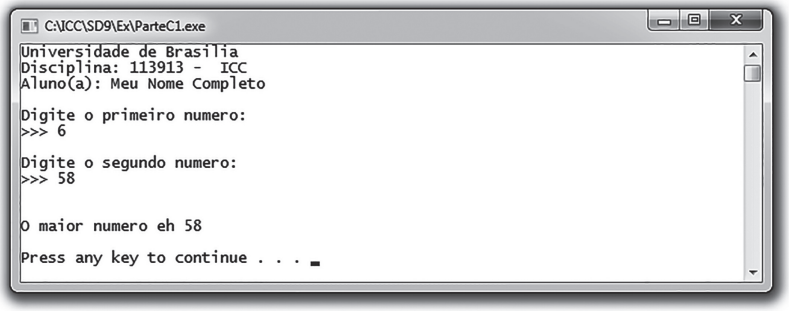
conversão dessa temperatura em Fahrenheit. Esse programa deve possuir uma função para converter a temperatura. Dados: Fahrenheit = Celsius x 1,8 + 32.

4. Escreva um programa que solicite dois números ao usuário e apresente na tela o resultado da soma do módulo desses números. Esse programa deve possuir uma função para calcular o módulo.
5. Desenvolva um programa que solicite três notas de provas de um aluno e apresente na tela a média dessas notas. A obtenção das notas deve ser feita na função principal, e o cálculo da média das notas deve ser obtido por outra função (MEDIA). Para o cálculo da média, considere que a primeira prova tem peso um, e as outras duas provas peso dois.
6. Escreva um programa que solicite dois números inteiros ao usuário e apresente na tela como resultado o dobro desses números, que devem ser somados e o resultado dessa soma ser triplicado. Esse programa deve possuir uma função para dobrar o valor de um número, outra para somar dois números e uma terceira para triplicar um número.
7. Desenvolva um programa que solicite ao usuário a idade de três pessoas e apresente na tela a maior delas. Esse programa deve possuir uma função para verificar qual é a maior idade.
8. Desenvolva um programa que solicite ao usuário a idade de três pessoas e apresente na tela a maior idade e a menor idade. Esse programa deve possuir uma função para verificar qual é a maior idade e outra para verificar a menor idade.
9. Escreva um programa que leia os dados de um funcionário e aplique um aumento sobre o seu

salário. A empresa definiu um aumento de 10% para quem possuir mais de cinco anos de casa e for casado; para o restante, o aumento é de 8%. Dados de entrada: salário bruto, quantidade de anos na empresa e estado civil (C, c, S, s). Dados de saída: salário inicial, taxa de aumento e salário com aumento.

Respostas das Atividades de Fixação

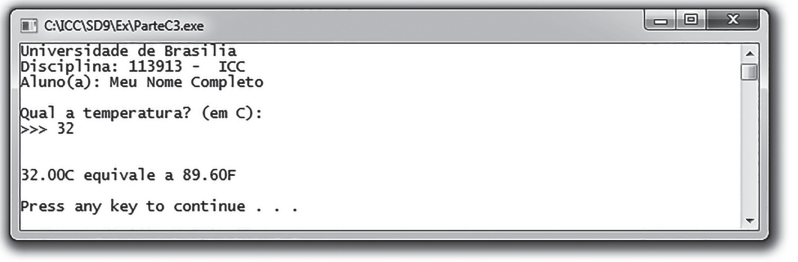
1)



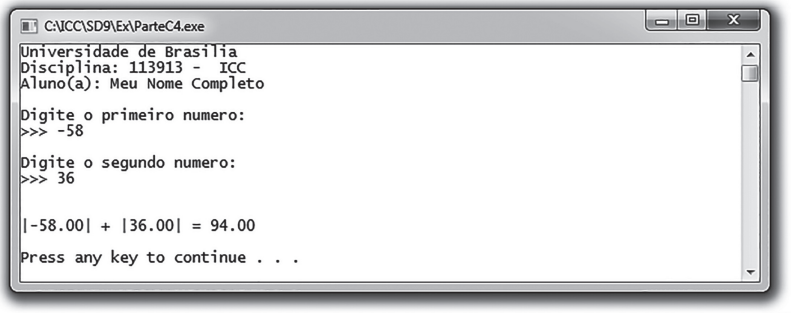
2)

A tela de resposta desta atividade é idêntica à mostrada no Capítulo 4, atividade 2.

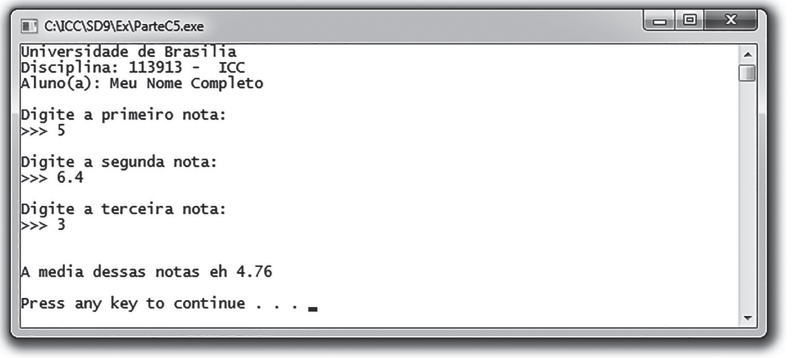
3)



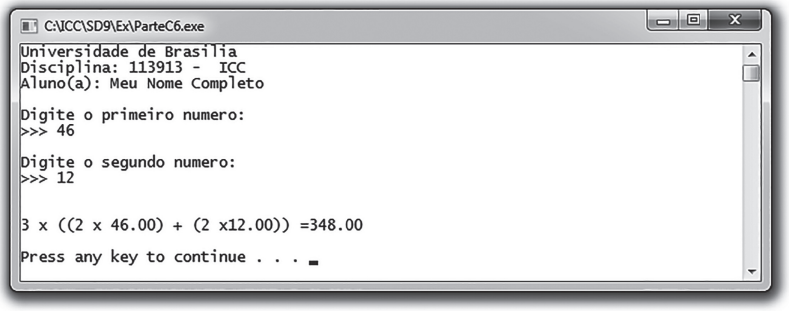
4)



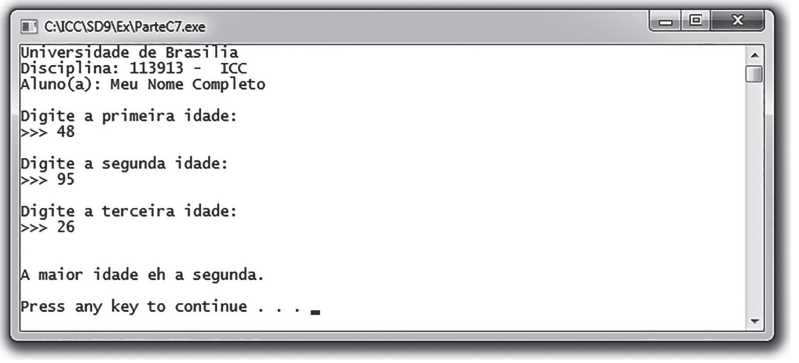
5)


C:\ICC\SD9\Ex\ParteC5.exe
Universidade de Brasilia
Disciplina: 113913 - ICC
Aluno(a): Meu Nome Completo
Digite a primeiro nota:
>>> 5
Digite a segunda nota:
>>> 6.4
Digite a terceira nota:
>>> 3
A media dessas notas eh 4.76
Press any key to continue . . .

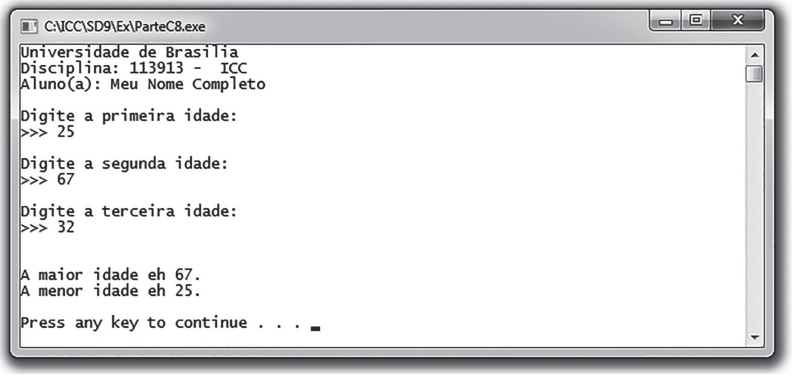
6)


C:\ICC\SD9\Ex\ParteC6.exe
Universidade de Brasilia
Disciplina: 113913 - ICC
Aluno(a): Meu Nome Completo
Digite o primeiro numero:
>>> 46
Digite o segundo numero:
>>> 12
3 x ((2 x 46.00) + (2 x12.00)) =348.00
Press any key to continue . . .

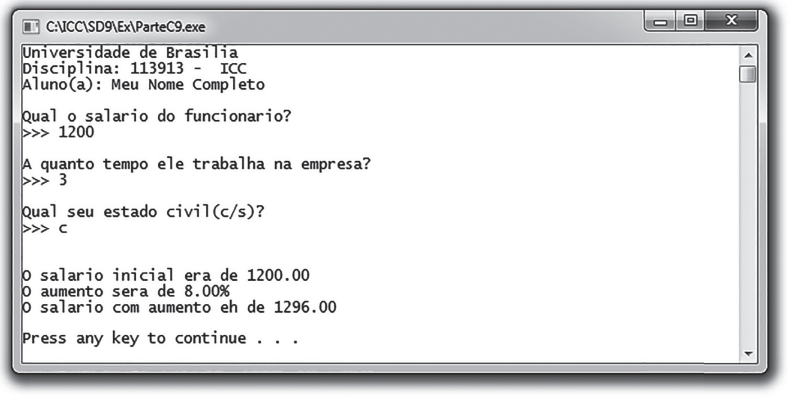
7)


C:\ICC\SD9\Ex\ParteC7.exe
Universidade de Brasilia
Disciplina: 113913 - ICC
Aluno(a): Meu Nome Completo
Digite a primeira idade:
>>> 48
Digite a segunda idade:
>>> 95
Digite a terceira idade:
>>> 26
A maior idade eh a segunda.
Press any key to continue . . .

8)


C:\ICC\SD9\Ex\ParteC8.exe
Universidade de Brasilia
Disciplina: 113913 - ICC
Aluno(a): Meu Nome Completo
Digite a primeira idade:
>>> 25
Digite a segunda idade:
>>> 67
Digite a terceira idade:
>>> 32
A maior idade eh 67.
A menor idade eh 25.
Press any key to continue . . .

9)


C:\ICC\SD9\Ex\ParteC9.exe
Universidade de Brasilia
Disciplina: 113913 - ICC
Aluno(a): Meu Nome Completo
Qual o salario do funcionario?
>>> 1200
A quanto tempo ele trabalha na empresa?
>>> 3
Qual seu estado civil(c/s)?
>>> c
O salario inicial era de 1200.00
O aumento sera de 8.00%
O salario com aumento eh de 1296.00
Press any key to continue . . .

Capítulo 10

Vetores

Vetores

Em programação de computadores, muitas vezes é necessário armazenar um conjunto de dados, geralmente da mesma ordem de grandeza e tipo. O vetor, também conhecido como array, é uma estrutura que pode conter um conjunto de elementos consecutivos, todos do mesmo tipo e que compartilham o mesmo nome. Um elemento específico em um vetor é acessado por meio de um índice, que, por sua vez, determina a posição de cada elemento. O índice geralmente é uma sequência de números inteiros.

A vantagem da utilização de vetores em programação é que os seus elementos são acessados de forma rápida. Na Figura 10.1, demonstramos a declaração de um vetor, ou matriz unidimensional.

```
tipo_do_vetor nome_vetor[tamanho do vetor]
```

a) Exemplo de declaração de vetor.

```
int notas[10] // vetor notas com 10 elementos
```

b) Exemplo de declaração de vetor em linguagem C.

Figura 10.1 – Exemplo de declaração de vetores na linguagem C

Fonte: Elaboração própria

O **<tipo_do_vetor>** corresponde ao tipo de dados de cada um dos elementos do vetor, que pode ser **float**, **int**, **double**, entre outros.

O **<nome_vetor[tamanho do vetor]>** indica o nome do vetor, por exemplo, alunos[quantidade de alunos], e, entre colchetes, o **<tamanho do vetor>** indica a quantidade de elementos a serem armazenados pelo vetor.

```
tipo_do_vetor nome_vetor[] = {elemento1,elemento2,....elementoN-1 };
```

a) Exemplo de declaração de vetor com inicialização.

```
int vetor[] = {0,1,2,3,4}; // vetor inicializado com 5 elementos
```

b) Exemplo de declaração de vetor com inicialização em linguagem C.

Figura 10.2 – Exemplo de declaração de vetores com inicialização do vetor na linguagem C

Fonte: Elaboração própria

Na Figura 10.2, apresentamos outra forma de declarar e inicializar o vetor. O **<tamanho do vetor>** não é informado e seu tamanho dependerá da quantidade de elementos que lhe são atribuídos na declaração.

```
int vetnotas[4] // vetor notas com 4 elementos

vetnotas[0] = 8;
vetnotas[1] = 7;
vetnotas[2] = 5;
vetnotas[3] = 6;
```

Figura 10.3 – Declaração do vetor vetnotas[4] de quatro posições e atribuição de valores ao vetnotas

Fonte: Elaboração própria

Na Figura 10.3, temos a declaração do vetor vetnotas[4] do tipo inteiro de quatro posições, e a atribuição de valores a cada posição do vetor utilizando o índice do vetor.

```
// laço para apresentar as notas do vetor vetnotas
for (auxcont=0; auxcont<4; auxcont++)
{
    printf ("\n\nA %d.a nota eh: %d: ", auxcont+1, vetnotas[auxcont]);
}
```

Figura 10.4 – Trecho de código para apresentar na tela os valores contidos no vetor vetnotas[ ] de quatro posições

Fonte: Elaboração própria

Na Figura 10.4, mostramos um trecho de código utilizando uma estrutura de repetição para apresentar os valores contidos no vetor vetnotas[4].

Experimento 10

Objetivo

O objetivo desse experimento é fazer você praticar a utilização de vetores na linguagem de programação C.

Orientações

Nesse experimento, você deve fazer o programa *Exemplovetor.cpp* solicitar a entrada de 5 notas, de 0 até 10, de um aluno. Esse programa deve calcular e apresentar a média aritmética dessas notas.

Procedimentos

Para executar o experimento, realize este passo a passo:

1. Em um sistema operacional, execute um IDE.
2. Digite o código do programa *Exemplovetor.cpp* (Figura 10.5).
3. Altere no código do cabeçalho o seu nome como o aluno que escreveu o programa.
4. Verifique se o código foi digitado corretamente.
5. Compile o programa *Exemplovetor.cpp*.
6. Caso o programa apresente alguma mensagem de erro, verifique em qual linha, qual erro e corrija-o.
7. Caso o programa não apresente mensagem de erro, execute-o.
8. Compare a sequência dos comandos apresentados na tela com a sequência dos comandos do programa.

******* Início do Código do programa: *vet.cpp* *******

```
1  #include <stdio.h> // Biblioteca de entrada e saida
2  #include <stdlib.h> //Biblioteca para uso do comando system("pause")
3
4  // Procedimento Cabeçalho
5  void cabecalho (){
6      printf("Universidade de Brasilia \n");
7      printf("Disciplina: 113913 -  ICC \n");
8      printf("Aluno(a): Meu Nome Completo \n\n");
9  }
10
```

a) Inclusão das bibliotecas e declaração dos procedimento de cabeçalho.

```
11 // Protótipo da função para calcular a média
12 float calmedia (float nota[5]);
13
14 // Protótipo da função para validação da nota
15 float validanota(float nota);
16
```

b) Declaração dos protótipos de função de calmedia e validanota.

```
17  int main(){
18
19      // Declaração das variaveis
20      float mediaaluno, notaprova[5];
21      int auxcont;
22
23      // Chamada e Inclusão do procedimento cabeçalho
24      cabecalho();
25      // Escrevendo mensagem na tela
26      printf ("\n\nEste programa solicita 5 notas, de 0 ate 10,");
27      printf ("\nde um aluno e calcula a media aritmetica.\n\n");
28      // Laço para obter as notas
29      for (auxcont=0; auxcont<5; auxcont++)
30      {
31       printf ("\n\nInforme a %d%c nota do aluno: ", auxcont+1, 166 );
32       // Primeira pocisão do vetor sendo o zero
33       // As pocisões são: notaprova[0], notaprova[1], ..., notaprova[4]
34       scanf ("%f",&notaprova[auxcont]);
35       notaprova[auxcont] = validanota(notaprova[auxcont]);
36      }
37      // Cálculo da média
38      mediaaluno = calmedia(notaprova);
39      printf ("\n\nA media do aluno %c: %.2f  \n\n", 130, mediaaluno);
40      // Para o sistema visualizar o resultado
41      system("PAUSE");
42      return 0;
43  }
```

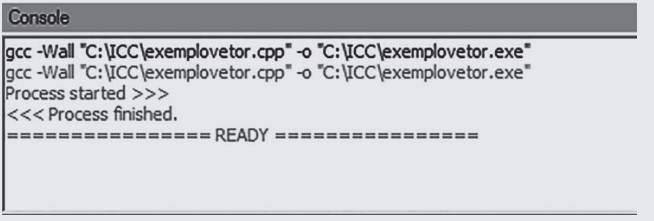
c) Função main() utilizando um vetor notaprova[ ] de 5 elementos.

```
44
45  // Definição do corpo da função para validação da nota
46  float validanota(float nota)
47  {
48      while ((nota < 0) || (nota > 10))
49      {
50          if (nota < 0)
51              printf("Voce informou uma nota menor que zero. \n");
52          else
53              printf("Voce informou uma nota maior que 10. \n");
54
55          printf("Digite novamente a nota: ");
56          scanf("%f", &nota);
57      }
58
59      return nota;
60  }
61
```

d) Função valida nota dos alunos.

```
62  // Definição do corpo da função para calcular a media
63  float calmedia (float nota[5])
64      {
65          float media = 0;
66          int cont;
67
68          for (cont=0; cont<5; cont++)
69              media = nota[cont] + media;
70
71          media /= 5;
72          return media;
73      }
```

e) Função calcula média.



f) Compilação do programa *Exemplovetor.cpp*.

******* Fim código original *******

Figura 10.5 – Código do Programa: *Exemplovetor.cpp*

Fonte: Elaboração própria

Na linha 35 da Figura 10.5, temos a validação da nota digitada. O vetor **notaprova**, na posição **auxcont**, recebe o resultado da função **validanota**, com o parâmetro da nota digitado no vetor.

Análise

Apresentamos o resultado da execução do programa *Exemplovetor.cpp* na Figura 10.6. Observe que o programa pede ao usuário as 5 primeiras notas de um aluno, que serão armazenadas em um vetor. Depois de digitar a última nota, o resultado da média aritmética dessas notas é apresentado na tela.

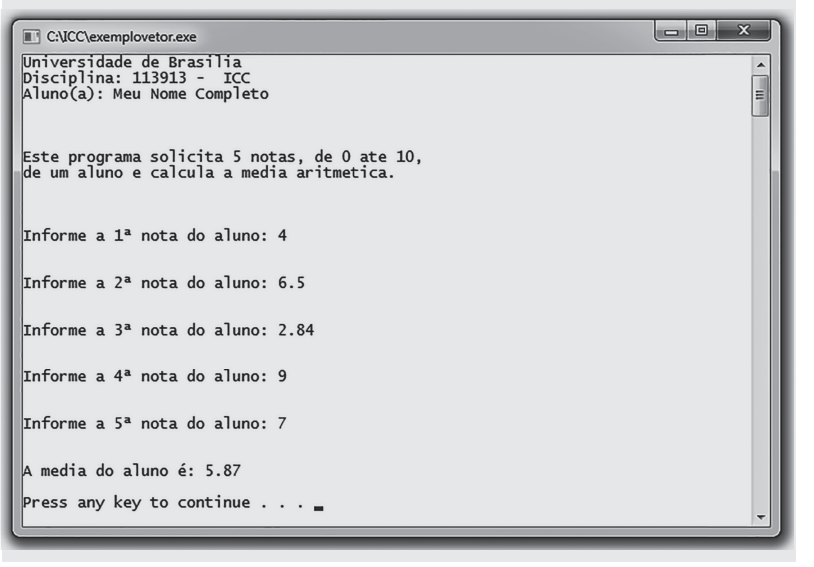


Figura 10.6 – Execução do programa: *Exemplovetor.cpp*

Fonte: Elaboração própria

Atividades de Fixação

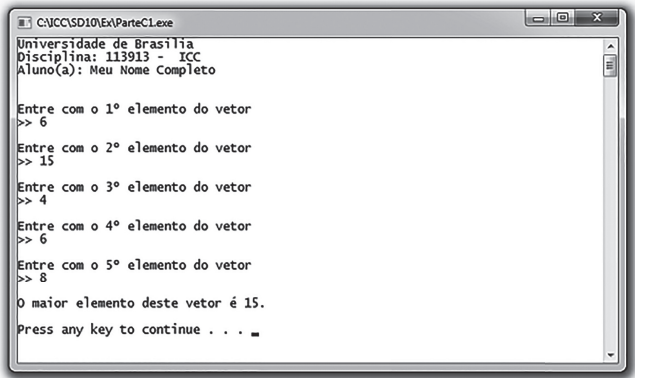
Para a resolução destas atividades de fixação, você deve:

- invocar o procedimento de cabeçalho;
- possuir mensagens que documentem o código;
- estar devidamente endentado;
- escrever as mensagens adequadas ao usuário para utilizar o programa;
- quando possível, utilizar a estrutura condicional composta;
- quando possível, utilizar a estrutura do tipo vetor;
- quando possível, utilizar estrutura de repetição; e
- quando possível, validar a entrada dos dados.

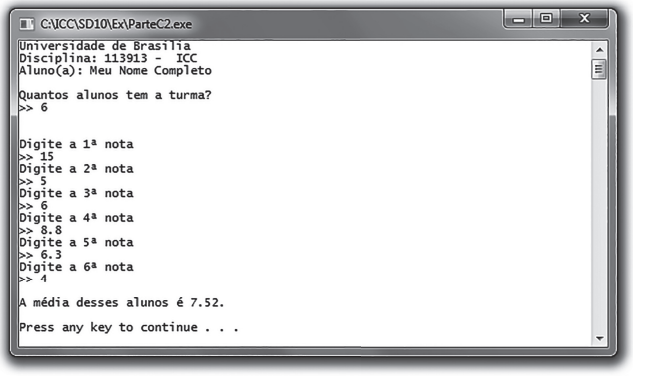
1. Escreva um programa para ler e informar o maior elemento de um vetor de 5 posições do tipo inteiro. Esse programa deve possuir uma função para verificar o maior número desse vetor.
2. Desenvolva um programa que calcule a média das notas de alunos para uma turma de no máximo 100 alunos. No início, faça o programa solicitar o tamanho da turma.
3. Escreva um programa que solicite as notas de 4 alunos ao usuário e apresente na tela a menor e a maior nota dos 4 alunos. Faça esse programa ter um procedimento de cabeçalho e uma função para verificar a menor e a maior nota dos alunos com um vetor de entrada NotAlunos[4].
4. Escreva um programa que solicite seis números do tipo inteiro ao usuário e os armazene em um vetor. Depois, faça o programa apresentar na tela os números na ordem inversa àquela na qual foram digitados.
5. Escreva um programa que leia 12 números do tipo inteiro ao usuário. Separe os números pares e ímpares lidos em dois outros vetores chamados **vetpar** e **vetimpar**. Em seguida, faça o programa apresentar os resultados na tela.

Respostas das Atividades de Fixação

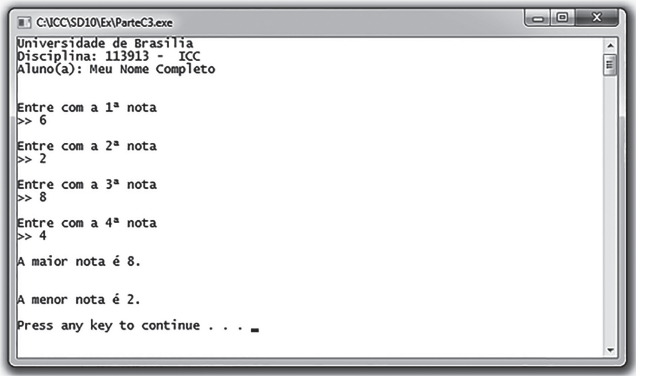
1)



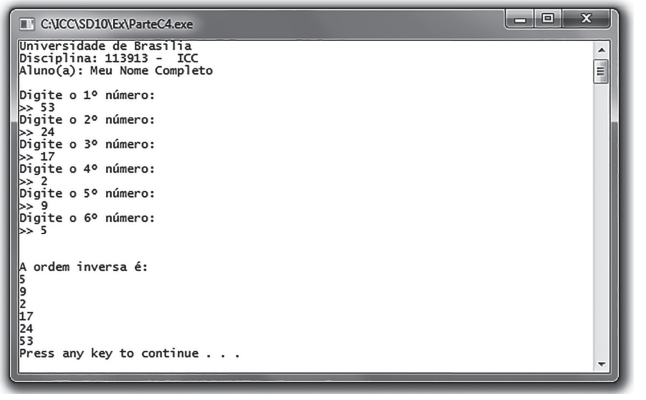
2)



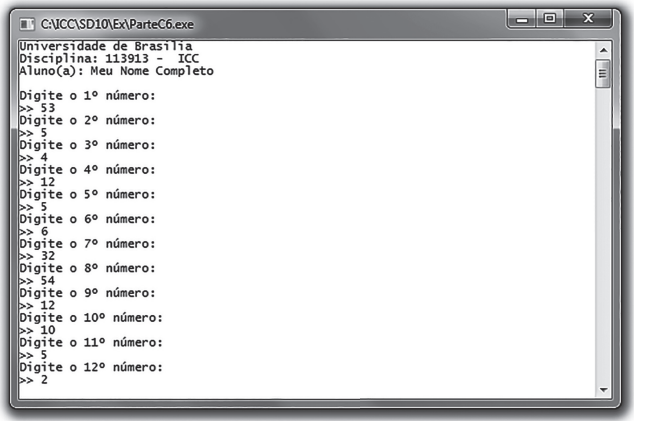
3)

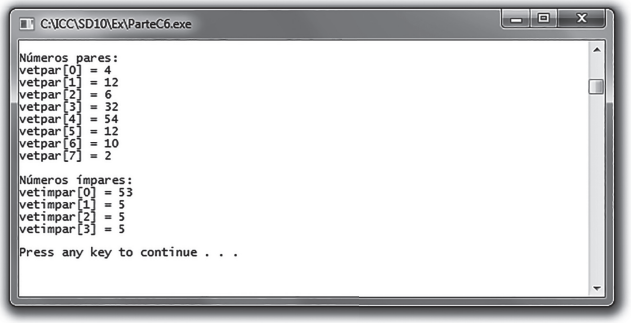


4)


C:\ICC\SD10\Ex\ParteC4.exe
Universidade de Brasília
Disciplina: 113913 - ICC
Aluno(a): Meu Nome Completo
Digite o 1º número:
>> 53
Digite o 2º número:
>> 24
Digite o 3º número:
>> 17
Digite o 4º número:
>> 2
Digite o 5º número:
>> 9
Digite o 6º número:
>> 5
A ordem inversa é:
5
9
2
17
24
53
Press any key to continue . . .

5)


C:\ICC\SD10\Ex\ParteC6.exe
Universidade de Brasília
Disciplina: 113913 - ICC
Aluno(a): Meu Nome Completo
Digite o 1º número:
>> 53
Digite o 2º número:
>> 5
Digite o 3º número:
>> 4
Digite o 4º número:
>> 12
Digite o 5º número:
>> 5
Digite o 6º número:
>> 6
Digite o 7º número:
>> 32
Digite o 8º número:
>> 54
Digite o 9º número:
>> 12
Digite o 10º número:
>> 10
Digite o 11º número:
>> 5
Digite o 12º número:
>> 2


C:\ICC\SD10\Ex\ParteC6.exe
Números pares:
vetpar[0] = 4
vetpar[1] = 12
vetpar[2] = 6
vetpar[3] = 32
vetpar[4] = 54
vetpar[5] = 12
vetpar[6] = 10
vetpar[7] = 2
Números ímpares:
vetimpar[0] = 53
vetimpar[1] = 5
vetimpar[2] = 5
vetimpar[3] = 5
Press any key to continue . . .

Capítulo 11

Matrizes

Matrizes

O vetor é uma estrutura que pode conter um conjunto de elementos consecutivos, todos do mesmo tipo e compartilhando o mesmo nome. A matriz é um vetor com mais de uma dimensão, ou seja, multidimensional. Também é um agrupamento de vetores unidimensionais.

A posição de cada elemento da matriz é determinada por índices. Na Figura 11.1, demonstramos a declaração de uma matriz.

```
tipo_matriz nome_matriz[tamanho dim1][tamanho dim2]...[tamanho dimN]
```

a) Exemplo de declaração de uma matriz com N dimensões.

```
tipo_matriz nome_matriz[nº de linhas][nº de colunas]
```

b) Exemplo de declaração de uma matriz com 2 dimensões.

```
// Declaração da matriz LadRetangulo de 4 por 2
float LadRetangulo[4][2];
```

c) Exemplo de declaração de uma matriz com 2 dimensões em linguagem C.

Figura 11.1 – Exemplos de declaração de matriz na linguagem C

Fonte: Elaboração própria

O **<tipo_matriz>** corresponde ao tipo de dados de cada um dos elementos na matriz e pode ser **float**, **int**, **double**, entre outros.

Para a matriz bidimensional, o **<nome_matriz[nro linhas] [nro colunas]>** indica o nome da matriz, por exemplo, LadRetangulo [4][2], e, dentro dos colchetes, o tamanho

da matriz, sendo o número dentro do primeiro colchete o indicador de quantidade de linhas e, no segundo, o indicador de quantidade de colunas da matriz.

LadRetangulo[4][2]

↓ \ →	0	1
0	2.2	4.5
1	2.8	4.5
2	8.5	3.1
3	7.6	7.1

a) Exemplo de matriz LadRetangulo com valores.

```
// laço para obter a BASE e ALTURA de cada RETANGULO
for (ContColuna=0; ContColuna<2; ContColuna++)
{
 printf ("\n\nInforme o valor da base de cada retangulo: \n", ContColuna+1 );
 // laço para obter os valores de cada RETANGULO
 for (ContLinha=0; ContLinha<4; ContLinha++)
 {
  printf ("Digite %d.o VALOR: ", ContLinha+1 );
  scanf ("%f",&ladretan[ContLinha][ContColuna]);
  // funcao para validar os valores recebidos
  ladretan[ContLinha][ContColuna] = validaladretan(ladretan[ContLinha][ContColuna]);
 }
}
```

b) Exemplo de trecho de código.

Figura 11.2 – Exemplo da matriz LadRetangulo com valores e trecho de código

Fonte: Elaboração própria

Na Figura 11.2, temos um exemplo gráfico da matriz LadRetangulo com quatro linhas e duas colunas: a primeira coluna representa os valores da base e a segunda coluna, os valores da altura do retângulo. Cada linha representa um retângulo, exemplificando as informações de quatro retângulos.

```
// laço para obter as notas dos 5 alunos
for (ContColuna=0; ContColuna<5; ContColuna++)
{
 printf ("\n\nInforme as  quatro notas do %d.o aluno: \n", ContColuna+1 );
 // laço para obter as 4 notas dos 5 alunos
 for (ContLinha=0; ContLinha<4; ContLinha++)
 {
  printf ("Digite %d.a nota: ", ContLinha+1 );
  scanf ("%f",&notaprova[ContLinha][ContColuna]);
  // funcao para validar a nota do aluno
  notaprova[ContLinha][ContColuna] = validanota(notaprova[ContLinha][ContColuna]);
 }
}
```

Figura 11.3 – Trecho de programa que obtém dados para uma matriz, notaprova, utilizando duas estruturas de repetição

Fonte: Elaboração própria

A Figura 11.3 apresenta um pedaço de um código que obtém dados para preencher uma matriz utilizando duas estruturas de repetição. A primeira estrutura de repetição é usada para manusear as colunas da matriz, e a estrutura de repetição interna para manusear as linhas da matriz.

Experimento 11

Objetivo

O objetivo desse experimento é ensinar você a utilizar matrizes na linguagem de programação C.

Orientações

Nesse experimento, você deve fazer o programa *Exemplomatriz.cpp* solicitar a base e a altura de quatro retângulos e armazenar os dados em uma matriz. A validação da entrada de dados consiste em verificar o tamanho da base e da altura que devem ser sempre maiores que zero. Em seguida, utilizando uma função, deve calcular o valor da área de cada retângulo

e apresentar na tela os lados e a área de cada retângulo armazenados na matriz.

Procedimentos

Para executar o experimento, realize este passo a passo:

1. Em um sistema operacional, execute um IDE.
2. Digite o código do programa *Exemplomatriz.cpp* (Figura 11.4).
3. Insira no código do cabeçalho o seu nome como nome do aluno que escreveu o programa.
4. Verifique se o código foi digitado corretamente.
5. Compile o programa *Exemplomatriz.cpp*.
6. Caso o programa apresente alguma mensagem de erro, verifique em qual linha, qual erro e corrija-o.
7. Caso o programa não apresente mensagem de erro, execute-o.
8. Compare a sequência dos comandos apresentados na tela com a sequência dos comandos do programa.

******* Início do Código do programa: *exemplomatriz.cpp* *******

```
1  #include <stdio.h> // Biblioteca de entrada e saida
2  #include <stdlib.h> //Biblioteca para uso do comando system("pause")
3
4  // Procedimento Cabeçalho
5  void cabecalho (){
6      printf("Universidade de Brasilia \n");
7      printf("Disciplina: 113913 -  ICC \n");
8      printf("Aluno(a): Meu Nome Completo \n\n");
9  }
10
```

a) Inclusão das bibliotecas e declaração do procedimento de cabeçalho.

```
11 // Protótipo da função para validacao do lado
12 int validalado(int lado);
13
14 // Protótipo da função para calcular a area do retangulo
15 int calcarea(int base, int altura);
16
```

b) Declaração dos protótipos das funções: validalado() e calcarea().

```
17 int main(){
18
19     // Declaração das variaveis
20     int LadRetangulo[3][4];
21     /* A matriz LadRetangulo[3][4] possui 3 linhas e 4 colunas
22        Exemplo:
23        LadRetangulo[0][1] = base do retangulo 2
24        LadRetangulo[1][1] = altura do retangulo 2
25        LadRetangulo[2][1] = area do retangulo 2
26     */
27     int ContColuna, ContLinha, AuxArea;
28     /* ContColuna variavel para manuseiar as colunas
29        ContLinha variavel para manuseiar as linhas
30     */
31     // Chamada e Inclusão do procedimento cabeçalho
32     cabecalho();
33     // Escrevendo mensagem na tela
34     printf ("\n\nEste programa solicita od ladod de 4 retangulos,");
35     printf ("\ndepois calcula a area dos retangulos.\n");
36
```

c) Função main() utilizando uma matriz LadRetangulo[3][4] de 3 linhas e 4 colunas.

```
37      // Laço para obter os lados dos retangulos
38      for (ContColuna=0; ContColuna<4; ContColuna++)
39      {
40          // Escrevendo mensagem na tela
41          printf ("\nInforme os lados do %d%c ret%cngulo: ", ContColuna+1, 167, 131);
42          printf ("\nInforme a base: ");
43          // Entrada dos dados para formação dos retângulos
44          scanf ("%d",&LadRetangulo[0][ContColuna]);
45          LadRetangulo[0][ContColuna] = validalado(LadRetangulo[0][ContColuna]);
46          // Escrevendo mensagem na tela
47          printf ("Informe a altura: ");
48          // Entrada dos dados para formação dos retângulos
49          scanf ("%d",&LadRetangulo[1][ContColuna]);
50          LadRetangulo[1][ContColuna] = validalado(LadRetangulo[1][ContColuna]);
51          // Cálculo da Área
52          AuxArea = calcarea(LadRetangulo[0][ContColuna],LadRetangulo[1][ContColuna]);
53          LadRetangulo[2][ContColuna] = AuxArea;
54  }
55
```

d) O código possui um laço para manusear a matriz na obtenção da base e altura de cada retângulo. O código faz chamada a duas funções: validalado() e calcarea().

```
56  // Laço para imprimir os lados e area dos retângulos
57  for (ContColuna=0; ContColuna<4; ContColuna++)
58  {
59    printf ("\n\nLados e area do %d%c ret%cngulo: ", ContColuna+1, 167, 131);
60    for (ContLinha=0; ContLinha<3; ContLinha++)
61    {
62     // primeira pocisão do vertor é o zero
63     if (ContLinha ==0)
64       printf ("\nBase: %d", LadRetangulo[ContLinha][ContColuna]);
65     else if (ContLinha ==1)
66             printf ("\nAltura: %d", LadRetangulo[ContLinha][ContColuna]);
67          else
68             printf ("\nArea: %d", LadRetangulo[ContLinha][ContColuna]);
69     }
70  }
71    printf("\n\n");
72    // Para o sistema visualizar o resultado
73    system("PAUSE");
74    return 0;
75  }
```

e) O código possui um laço para manusear a matriz e apresentar seus dados. São mostrados na tela os valores da base, da altura e da área de cada retângulo.

```
77  // Definição do corpo da função para validação do lado
78  int validalado(int lado)
79  {
80      while (lado <= 0)
81      {
82          if (lado == 0)
83             printf("\nVoce informou um lado igual a zero. \n");
84          else
85             printf("\nVoce informou um lado menor que zero. \n");
86
87          printf("Digite novamente a lado sendo maior que zero: ");
88          scanf("%d", &lado);
89      }
90      return lado;
91  }
92
```

f) Código da função validalado().

```
93  // Definição do corpo da função para calcular a área do retângulo
94  int calcarea(int base, int altura)
95      {
96          int area;
97          area = base*altura;
98          return area;
99      }
```

g) Código da função calcarea().

```
Console
gcc -Wall "C:\ICC\exemplomatriz.cpp" -o "C:\ICC\exemplomatriz.exe"
gcc -Wall "C:\ICC\exemplomatriz.cpp" -o "C:\ICC\exemplomatriz.exe"
Process started >>>
<<< Process finished.
================ READY ================
```

h) Compilação do programa *Exemplomatriz.cpp*.

******* Fim código original *******

Figura 11.4 – Código do Programa: *Exemplomatriz.cpp*

Fonte: Elaboração própria

Análise

Apresentamos o resultado da execução do programa *Exemplomatriz.cpp* na Figura 11.5.

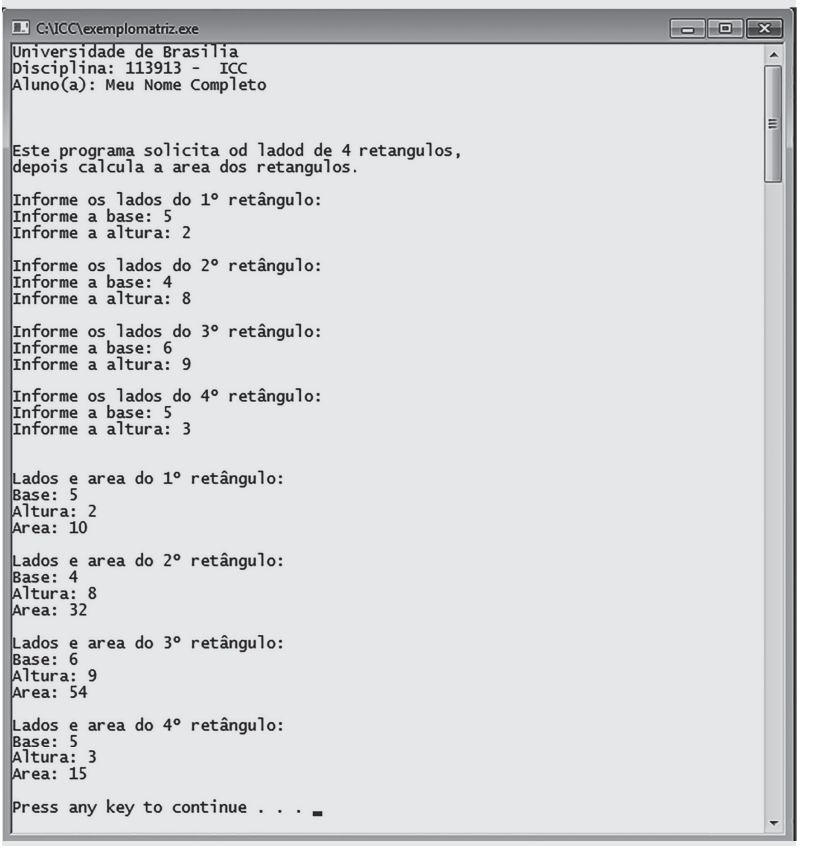


Figura 11.5 – Execução do programa: *Exemplomatriz.cpp*

Fonte: Elaboração própria

Atividades de Fixação

Para a resolução destas atividades de fixação, você deve:

- invocar o procedimento de cabeçalho;
- possuir mensagens que documentem o código;
- estar devidamente endentado;
- escrever as mensagens adequadas ao usuário para utilizar o programa;
- quando possível, utilizar a estrutura condicional composta;
- quando possível, validar a entrada dos dados; e
- quando possível, utilizar estrutura de repetição.

1. Faça um programa para obter os valores de uma matriz 6x6 de números inteiros. Depois da leitura dos dados, faça o programa calcular a soma dos elementos da diagonal principal e mostrar os valores da matriz e da soma.
2. Desenvolva um programa para ler os dados de uma matriz que armazene três notas de dez alunos. Em seguida, faça o programa apresentar a menor nota da prova de cada aluno. Desenvolva, ainda, uma função para obter a menor nota de cada aluno.
3. Desenvolva um programa com uma matriz notaprova para armazenar quatro notas de cinco alunos e, depois, apresente na tela essas notas. Esse programa dever possuir validação de dados de entrada. Utilize duas estruturas de repetição: uma para manusear a coluna da matriz; outra, para manusear as linhas. O resultado do programa deve corresponder aos das Figuras 11.6 e 11.7, que representam a obtenção dos dados e a apresentação dos dados na tela, respectivamente.

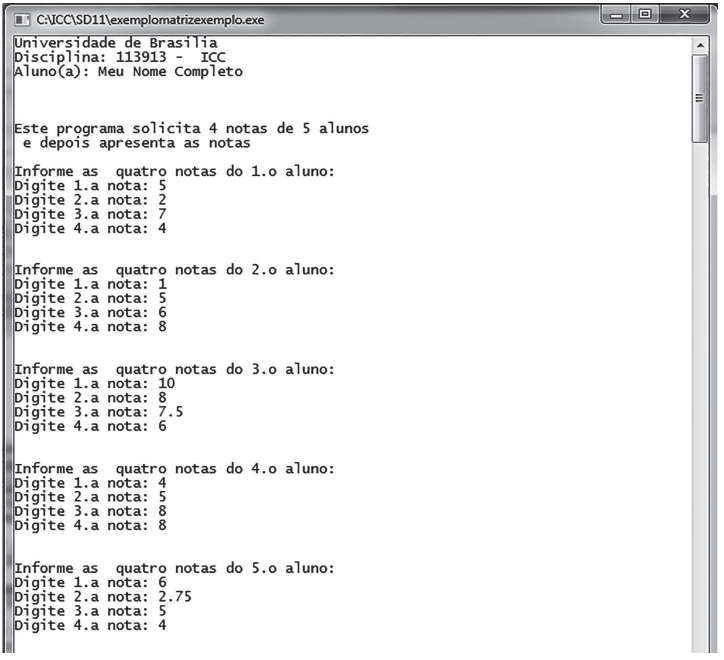


Figura 11.6 – Trecho do resultado da execução do código do programa em C a ser elaborado que obtém as notas de alunos

Fonte: Elaboração própria

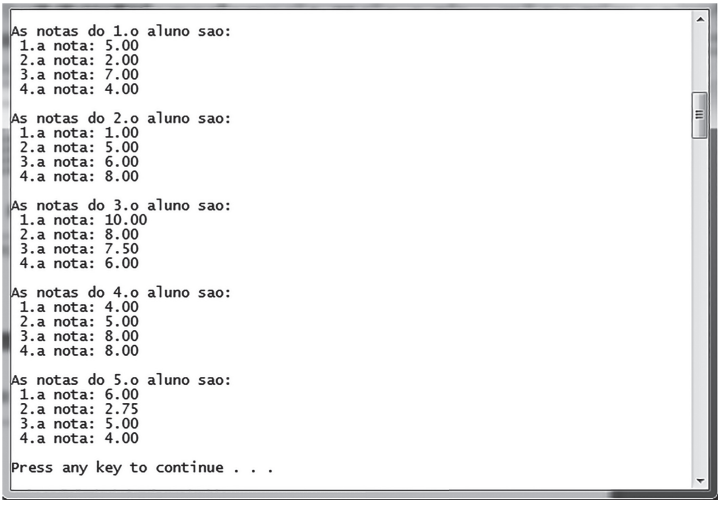


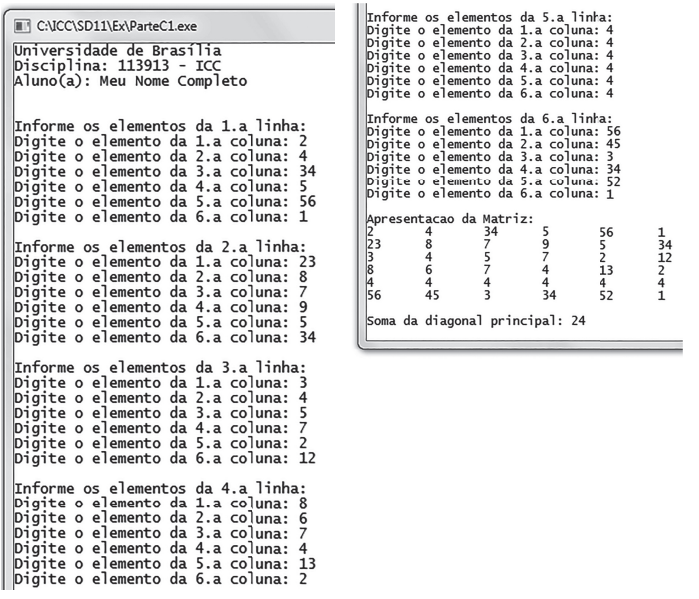
Figura 11.7 – Trecho do resultado da execução do código do programa em C a ser elaborado que apresenta as notas dos alunos

Fonte: Elaboração própria

4. Desenvolva um programa com uma matriz 4x8 de números inteiros e some cada uma das linhas da matriz guardando o resultado da soma em um vetorsomalinha. Em seguida, apresente os valores da matriz e do vetor.

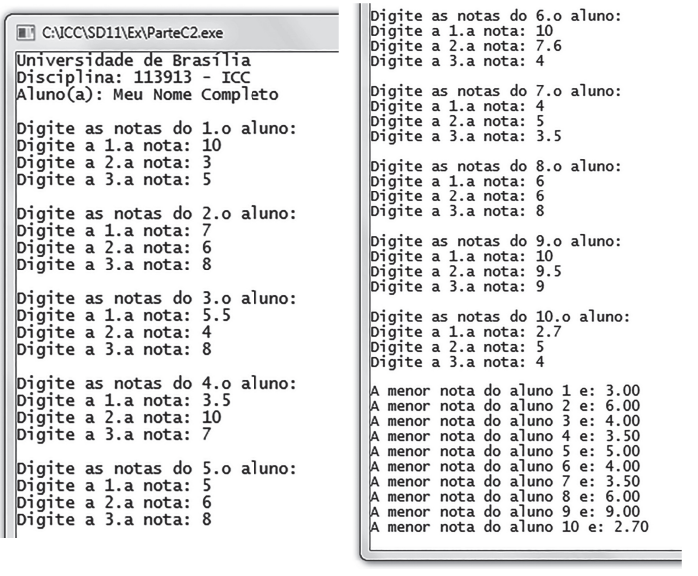
Respostas das Atividades de Fixação

1)



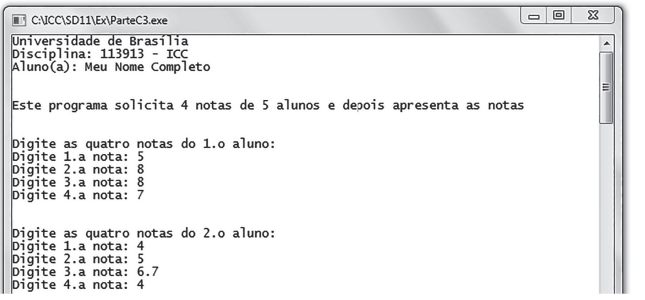
```
C:\ICC\SD11\Ex\ParteC1.exe
Universidade de Brasília
Disciplina: 113913 - ICC
Aluno(a): Meu Nome Completo

Informe os elementos da 1.a linha:
Digite o elemento da 1.a coluna: 2
Digite o elemento da 2.a coluna: 4
Digite o elemento da 3.a coluna: 34
Digite o elemento da 4.a coluna: 5
Digite o elemento da 5.a coluna: 56
Digite o elemento da 6.a coluna: 1

Informe os elementos da 2.a linha:
Digite o elemento da 1.a coluna: 23
Digite o elemento da 2.a coluna: 8
Digite o elemento da 3.a coluna: 7
Digite o elemento da 4.a coluna: 9
Digite o elemento da 5.a coluna: 5
Digite o elemento da 6.a coluna: 34

Informe os elementos da 3.a linha:
Digite o elemento da 1.a coluna: 3
Digite o elemento da 2.a coluna: 4
Digite o elemento da 3.a coluna: 5
Digite o elemento da 4.a coluna: 7
Digite o elemento da 5.a coluna: 2
Digite o elemento da 6.a coluna: 12

Informe os elementos da 4.a linha:
Digite o elemento da 1.a coluna: 8
Digite o elemento da 2.a coluna: 6
Digite o elemento da 3.a coluna: 7
Digite o elemento da 4.a coluna: 4
Digite o elemento da 5.a coluna: 13
Digite o elemento da 6.a coluna: 2

Informe os elementos da 5.a linha:
Digite o elemento da 1.a coluna: 4
Digite o elemento da 2.a coluna: 4
Digite o elemento da 3.a coluna: 4
Digite o elemento da 4.a coluna: 4
Digite o elemento da 5.a coluna: 4
Digite o elemento da 6.a coluna: 4

Informe os elementos da 6.a linha:
Digite o elemento da 1.a coluna: 56
Digite o elemento da 2.a coluna: 45
Digite o elemento da 3.a coluna: 3
Digite o elemento da 4.a coluna: 34
Digite o elemento da 5.a coluna: 52
Digite o elemento da 6.a coluna: 1

Apresentacao da Matriz:
2       4       34      5       56      1
23      8       7       9       5       34
3       4       5       7       2       12
8       6       7       4       13      2
4       4       4       4       4       4
56      45      3       34      52      1

Soma da diagonal principal: 24
```

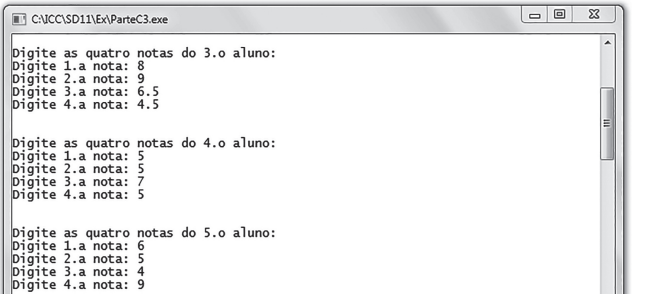
2)

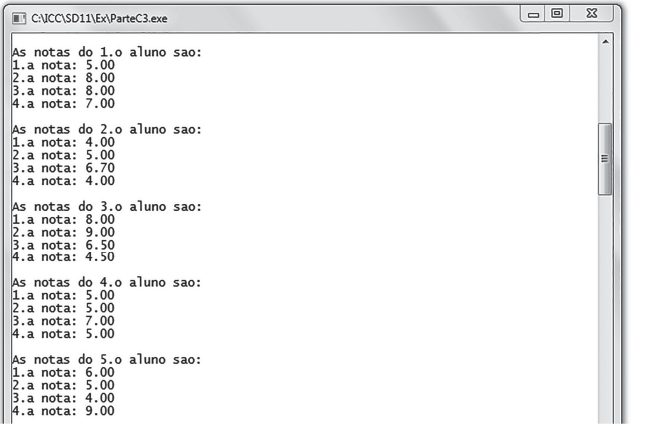


```
C:\ICC\SD11\Ex\ParteC2.exe
Universidade de Brasília
Disciplina: 113913 - ICC
Aluno(a): Meu Nome Completo

Digite as notas do 1.o aluno:
Digite a 1.a nota: 10
Digite a 2.a nota: 3
Digite a 3.a nota: 5

Digite as notas do 2.o aluno:
Digite a 1.a nota: 7
Digite a 2.a nota: 6
Digite a 3.a nota: 8

Digite as notas do 3.o aluno:
Digite a 1.a nota: 5.5
Digite a 2.a nota: 4
Digite a 3.a nota: 8

Digite as notas do 4.o aluno:
Digite a 1.a nota: 3.5
Digite a 2.a nota: 10
Digite a 3.a nota: 7

Digite as notas do 5.o aluno:
Digite a 1.a nota: 5
Digite a 2.a nota: 6
Digite a 3.a nota: 8

Digite as notas do 6.o aluno:
Digite a 1.a nota: 10
Digite a 2.a nota: 7.6
Digite a 3.a nota: 4

Digite as notas do 7.o aluno:
Digite a 1.a nota: 4
Digite a 2.a nota: 5
Digite a 3.a nota: 3.5

Digite as notas do 8.o aluno:
Digite a 1.a nota: 6
Digite a 2.a nota: 6
Digite a 3.a nota: 8

Digite as notas do 9.o aluno:
Digite a 1.a nota: 10
Digite a 2.a nota: 9.5
Digite a 3.a nota: 9

Digite as notas do 10.o aluno:
Digite a 1.a nota: 2.7
Digite a 2.a nota: 5
Digite a 3.a nota: 4

A menor nota do aluno 1 e: 3.00
A menor nota do aluno 2 e: 6.00
A menor nota do aluno 3 e: 4.00
A menor nota do aluno 4 e: 3.50
A menor nota do aluno 5 e: 5.00
A menor nota do aluno 6 e: 4.00
A menor nota do aluno 7 e: 3.50
A menor nota do aluno 8 e: 6.00
A menor nota do aluno 9 e: 9.00
A menor nota do aluno 10 e: 2.70
```

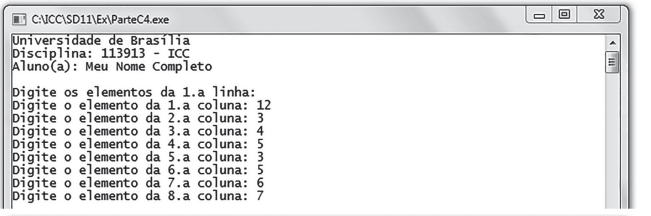
3)

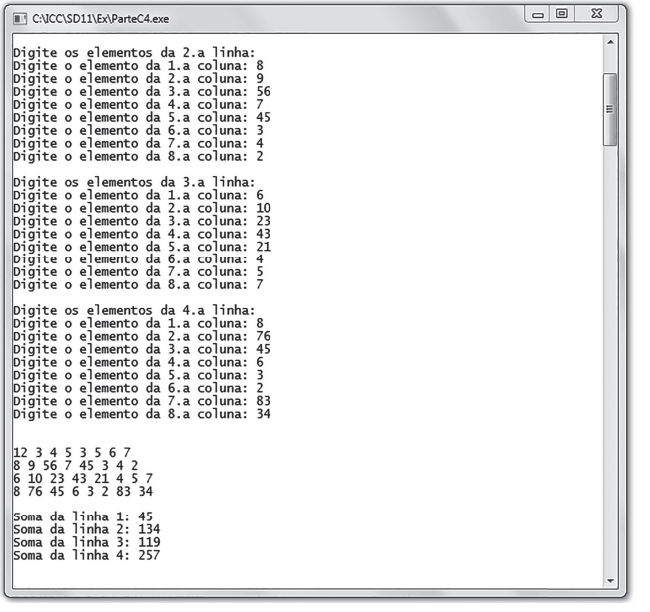

C:\ICC\SD11\Ex\ParteC3.exe
Universidade de Brasília
Disciplina: 113913 - ICC
Aluno(a): Meu Nome Completo
Este programa solicita 4 notas de 5 alunos e depois apresenta as notas
Digite as quatro notas do 1.o aluno:
Digite 1.a nota: 5
Digite 2.a nota: 8
Digite 3.a nota: 8
Digite 4.a nota: 7
Digite as quatro notas do 2.o aluno:
Digite 1.a nota: 4
Digite 2.a nota: 5
Digite 3.a nota: 6.7
Digite 4.a nota: 4


C:\ICC\SD11\Ex\ParteC3.exe
Digite as quatro notas do 3.o aluno:
Digite 1.a nota: 8
Digite 2.a nota: 9
Digite 3.a nota: 6.5
Digite 4.a nota: 4.5
Digite as quatro notas do 4.o aluno:
Digite 1.a nota: 5
Digite 2.a nota: 5
Digite 3.a nota: 7
Digite 4.a nota: 5
Digite as quatro notas do 5.o aluno:
Digite 1.a nota: 6
Digite 2.a nota: 5
Digite 3.a nota: 4
Digite 4.a nota: 9


C:\ICC\SD11\Ex\ParteC3.exe
As notas do 1.o aluno sao:
1.a nota: 5.00
2.a nota: 8.00
3.a nota: 8.00
4.a nota: 7.00
As notas do 2.o aluno sao:
1.a nota: 4.00
2.a nota: 5.00
3.a nota: 6.70
4.a nota: 4.00
As notas do 3.o aluno sao:
1.a nota: 8.00
2.a nota: 9.00
3.a nota: 6.50
4.a nota: 4.50
As notas do 4.o aluno sao:
1.a nota: 5.00
2.a nota: 5.00
3.a nota: 7.00
4.a nota: 5.00
As notas do 5.o aluno sao:
1.a nota: 6.00
2.a nota: 5.00
3.a nota: 4.00
4.a nota: 9.00

4)


C:\ICC\SD11\Ex\ParteC4.exe
Universidade de Brasília
Disciplina: 113913 - ICC
Aluno(a): Meu Nome Completo
Digite os elementos da 1.a linha:
Digite o elemento da 1.a coluna: 12
Digite o elemento da 2.a coluna: 3
Digite o elemento da 3.a coluna: 4
Digite o elemento da 4.a coluna: 5
Digite o elemento da 5.a coluna: 3
Digite o elemento da 6.a coluna: 5
Digite o elemento da 7.a coluna: 6
Digite o elemento da 8.a coluna: 7


C:\ICC\SD11\Ex\ParteC4.exe
Digite os elementos da 2.a linha:
Digite o elemento da 1.a coluna: 8
Digite o elemento da 2.a coluna: 9
Digite o elemento da 3.a coluna: 56
Digite o elemento da 4.a coluna: 7
Digite o elemento da 5.a coluna: 45
Digite o elemento da 6.a coluna: 3
Digite o elemento da 7.a coluna: 4
Digite o elemento da 8.a coluna: 2
Digite os elementos da 3.a linha:
Digite o elemento da 1.a coluna: 6
Digite o elemento da 2.a coluna: 10
Digite o elemento da 3.a coluna: 23
Digite o elemento da 4.a coluna: 43
Digite o elemento da 5.a coluna: 21
Digite o elemento da 6.a coluna: 4
Digite o elemento da 7.a coluna: 5
Digite o elemento da 8.a coluna: 7
Digite os elementos da 4.a linha:
Digite o elemento da 1.a coluna: 8
Digite o elemento da 2.a coluna: 76
Digite o elemento da 3.a coluna: 45
Digite o elemento da 4.a coluna: 6
Digite o elemento da 5.a coluna: 3
Digite o elemento da 6.a coluna: 2
Digite o elemento da 7.a coluna: 83
Digite o elemento da 8.a coluna: 34
12 3 4 5 3 5 6 7
8 9 56 7 45 3 4 2
6 10 23 43 21 4 5 7
8 76 45 6 3 2 83 34
Soma da linha 1: 45
Soma da linha 2: 134
Soma da linha 3: 119
Soma da linha 4: 257

Capítulo 12

Strings

Strings

Armazenar e manipular textos como palavras, nomes e sentenças são dois dos usos mais importantes dos vetores. Uma string é um conjunto de caracteres armazenados em um vetor sempre terminado pelo delimitador \0. Ou seja, uma string é um conjunto de caracteres consecutivos que ocupam um *byte* de memória armazenado em sequência e terminado por \0.

Na Figura 12.1, exemplificamos a declaração de uma string. O **<tipo_do_vetor>** para uma string sempre será do tipo char. O **<nome_vetor[quantidade de caracteres]>** indica o nome da string, por exemplo, NomeAluno[tamanho]. Entre colchetes, a quantidade de caracteres indica a quantidade de caracteres a serem armazenados pela string. Devemos sempre somar mais uma posição para armazenar o espaço do delimitador no final.

```
tipo_do_vetor nome_vetor[quantidade de caracteres]
```

a) Exemplo de declaração de uma string.

```
char NomeAluno[50]; // String Nome com 50 caracteres
```

b) Exemplo de declaração de *string* em linguagem C.

Figura 12.1 – Exemplo de declaração de string

Fonte: Elaboração própria

A Figura 12.2 apresenta algumas formas de declarar e inicializar uma string. Apenas na Figura 12.2 a), o tamanho da string é definido pelo valor informado na sua quantidade de

caracteres. Nos exemplos b) e c), o tamanho da stringNome-Aluno é de 18 posições, ou seja, 17+1, sendo 17 a quantidade de caracteres atribuídos à string e o 1 o tamanho do delimitador, '\0', no final da string.

```
char NomeAluno[10]={'N','o','m','e','A','l','u','n','o'};
```

a) **Exemplo de declaração de uma string com atribuição.**

```
char NomeAluno[] = "Meu Nome Completo";
```

b) Exemplo de declaração de uma string com o tamanho definido pela atribuição do conteúdo.

```
char *NomeAluno = "Meu Nome Completo";
```

c) Exemplo de declaração de uma string com o tamanho definido pela atribuição do conteúdo.

Figura 12.2 – Exemplo de declaração de strings com inicialização na linguagem C

Fonte: Elaboração própria

Na Figura 12.3, usamos o formato "%s" para manusear o tipo string nas funções **scanf** ou **printf**.

```
printf("Nome do Aluno: ");
scanf("%s", NomeAluno);
```

Figura 12.3 – Exemplo do comando da função **printf** e **scanf**

Fonte: Elaboração própria

Na atribuição de uma string, a função **scanf** lê todos os caracteres até encontrar um término de entrada de dados: <ENTER> ou <ESPAÇO> ou <TAB>. Quando o término de uma leitura é realizado por <ESPAÇO> ou <TAB>, os demais caracteres depois desses dois símbolos serão armazenadas no *buffer* de memória.

Na função **scanf** para a leitura de um tipo string, a variável não deve ser precedida de um &, como podemos ver na variável NomeAluno na Figura 12.3.

Para armazenar mais de uma palavra em uma variável, podemos utilizar a função **gets** (**getstring**). Outra função para manipular strings é a função **puts**, que permite unicamente a escrita de strings na tela. A Figura 12.4 ilustra um exemplo de utilização das funções **puts** e **gets**.

```
puts("Nome do Aluno: ");
gets(NomeAluno);
```

Figura 12.4 – Exemplo do comando da função **puts** e **gets**

Fonte: Elaboração própria

A única diferença entre a função **puts** e a função **printf** é que a função **puts** força uma mudança de linha automaticamente.

```
// funcao para contar numero de caracteres
// de uma string
int strlen(char *auxstring)
{
  int cont = 0;
  while (auxstring[cont] != '\0')
    cont++;

  return cont;
}
```

Figura 12.5 – Trecho de código para apresentar a função strlen

Fonte: Elaboração própria

Na Figura 12.5, o trecho de um código da função **strlen** realiza a contagem e o retorno da quantidade de caracteres de uma string. A função recebe como parâmetro a string e, com um laço de repetição, percorre os elementos até encontrar o delimitador \0 do fim da string.

Experimento 12

Objetivo

O objetivo desse experimento é você praticar a utilização de strings na linguagem de programação C.

Orientações

Nesse experimento, você deve fazer o programa *Exemplostring.cpp* solicitar os nomes do curso e do aluno e validar se os dados foram preenchidos.

Procedimentos

Para executar o experimento, realize este passo a passo respeitando a sequência dada nas partes A e B:

Parte A

1. Em um sistema operacional, execute um IDE.
2. Digite o código do programa *Exemplostring.cpp* Figura 12.6).
3. Altere o código do cabeçalho para o seu nome como o aluno que escreveu o programa.
4. Verifique se o código foi digitado corretamente.
5. Compile o programa.
6. Caso o programa apresente alguma mensagem de erro, verifique em qual linha, qual erro e corrija-o.
7. Caso o programa não apresente mensagem de erro, execute-o.
8. Compare a sequência dos comandos apresentados na tela com a sequência dos comandos do programa.

Parte B

1. Altere o código do exercício anterior e salve como *contastring.cpp*. Esse programa deverá apresentar a quantidade de caracteres das strings, nome e curso do aluno.

***** **Início do Código do programa: *exemplostring.cpp*** *****

```
1  #include <stdio.h> // Biblioteca de entrada e saida
2  #include <stdlib.h> //Biblioteca para uso do comando system("pause")
3
4  void cabecalho() {
5      printf("Universidade de Brasilia \n");
6      printf("Disciplina: 113913 -  ICC \n");
7      printf("Aluno(a): Meu Nome Completo \n\n");
8  }
9
```

a) Inclusão das bibliotecas e declaração do procedimento cabeçalho.

```
10 //função principal
11 int main()
12 {
13  // declaração das variáveis
14  char NomeAluno[40]; // string NomeAluno de 40 caracteres
15  char NomeCurso[40]; // string NomeCurso de 40 caracteres
16
17  // Esse programa obtem o nome do aluno e do seu curso
18  // após apresenta-os na tela
19
20  cabecalho();
21
```

b) Início da função main() utilizando as *strings*NomeAluno[40] e NomeCurso[40] e invocação do cabeçalho.

```
21
22  printf("\nNome do aluno: "); gets(NomeAluno);
23
24  // Verifica se o nome foi digitado
25  while (NomeAluno[0] =='\0')
26  {
27   puts("Nome em branco. \nDigite o nome do aluno: ");
28   gets(NomeAluno);
29  }
```

c) Obtenção de dados para a *string*NomeAluno[] e sua validação.

```
30
31      puts(NomeAluno);
32
33     printf("Nome do curso: "); gets(NomeCurso);
34     // Verifica se o nome foi digitado
35     while (NomeCurso[0] =='\0')
36     {
37      puts("Nome em branco. \nDigite o nome do curso: ");
38      gets(NomeCurso);
39     }
```

d) Obtenção de dados para a *string*NomeCurso [] e sua validação.

```
40
41     printf("\n\nAs informacoes digitadas foram: \n");
42     printf("Nome do aluno: %s \n", NomeAluno);
43     printf("Nome do curso: %s \n\n", NomeCurso);
44
45        // Para o sistema visualizar o resultado
46        system("PAUSE");
47        return 0;
48  }
```

e) Apresentação dos dados das *strings*NomeAluno[] e NomeCurso[].

```
Console
gcc -Wall C:\ICC\SD12\exemplostring.cpp -o C:\ICC\SD12\exemplostring.exe
gcc -Wall C:\ICC\SD12\exemplostring.cpp -o C:\ICC\SD12\exemplostring.exe
Process started >>>
<<< Process finished.
================ READY ================
```

f) Compilação do programa *exemplostring.cpp*.

******* Fim código original *******

Figura 12.6 – Código do Programa: *exemplo*string.*cpp*

Fonte: Elaboração própria

Análise

Apresentamos o resultado da execução do programa *Exemplo*string.*cpp* na Figura 12.7. Observe que o programa

pede ao usuário o nome e o curso do aluno e, depois, mostra o resultado na tela. Também valida a entrada de dados, não podendo ser branco.

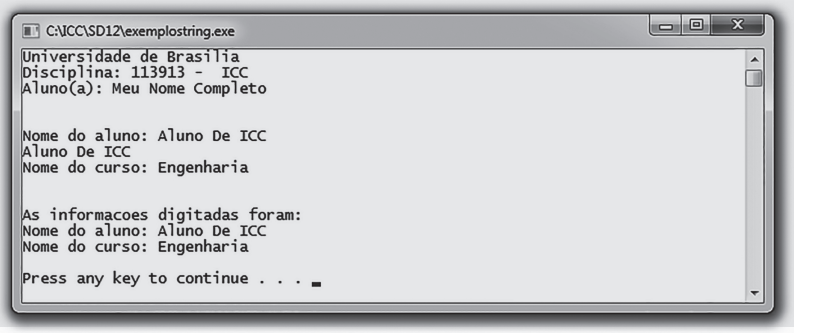


Figura 12.7 – Execução do programa: *Exemplo*string.*cpp*

Fonte: Elaboração própria

Apresentamos o resultado da execução da Parte B do procedimento do programa *Contastring.cpp* na Figura 12.8.

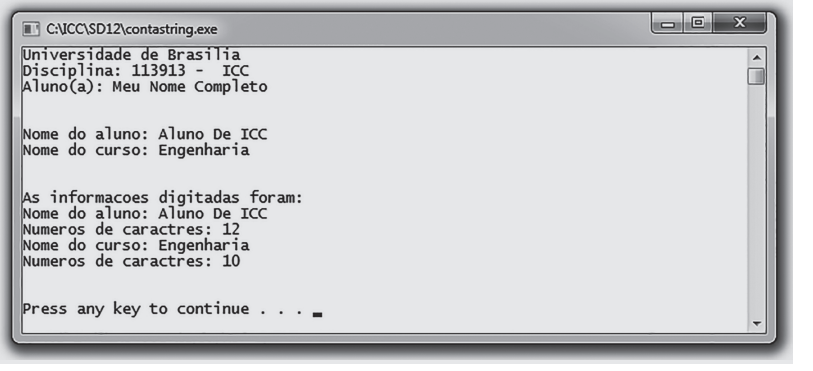


Figura 12.8 – Execução do programa: *Conta*string.*cpp*

Fonte: Elaboração própria

Note a quantidade de caracteres digitados em cada pergunta, observando que o espaço entre as palavras também é contado.

Atividades de Fixação

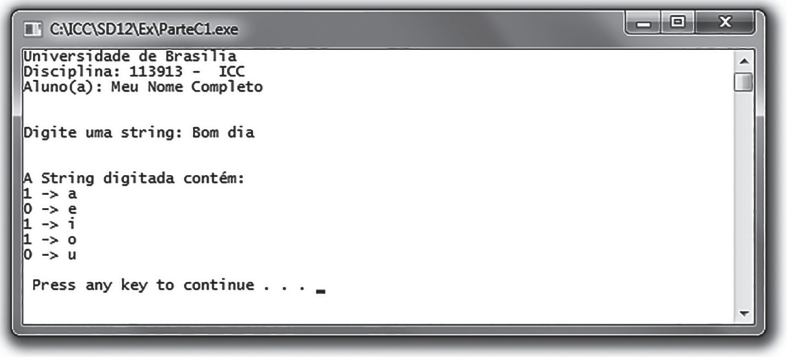
Para a resolução destas atividades de fixação, você deve:

- invocar o procedimento de cabeçalho;
- possuir mensagens que documentem o código;
- estar devidamente endentado;
- escrever as mensagens adequadas ao usuário para utilizar o programa;
- quando possível, utilizar a estrutura condicional composta;
- quando possível, validar a entrada dos dados; e
- quando possível, utilizar estrutura de repetição.

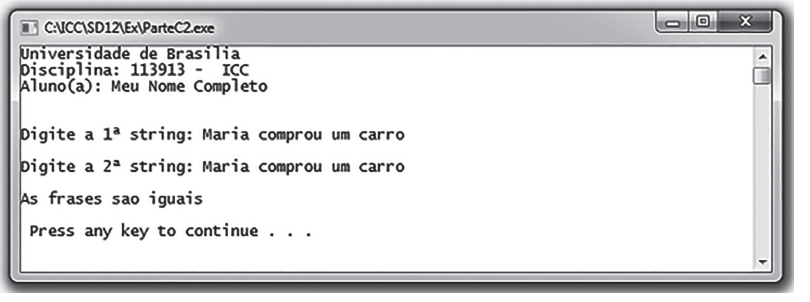
1. Desenvolva um programa para ler uma string (frase) com tamanho máximo de 100 caracteres. Faça o programa contar a quantidade de cada vogal que aparece na frase e, depois, informar ao usuário.
2. Desenvolva um programa que leia duas strings (frases) de tamanho máximo de 60 caracteres. Faça o programa verificar e informar se essas duas frases são iguais.
3. Desenvolva um programa que leia uma palavra de tamanho máximo de 60 caracteres. Faça o programa verificar e informar se essa palavra é um palíndromo. Uma palavra é palíndromo quando ela pode ser lida da mesma forma da esquerda para direita e da direita para esquerda.
4. Desenvolva um programa para ler uma palavra de no máximo 60 caracteres. Faça o programa escrever essa palavra de trás para frente.

Respostas das Atividades de Fixação

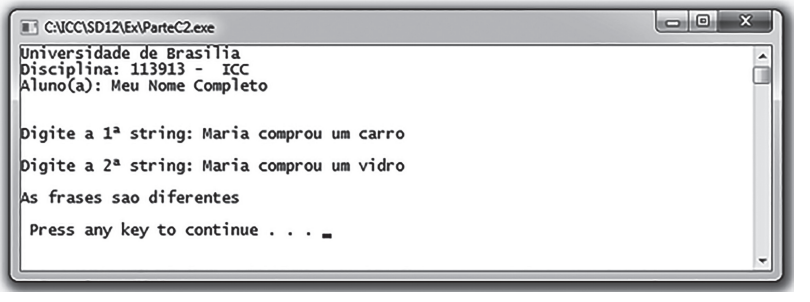
1)



2)

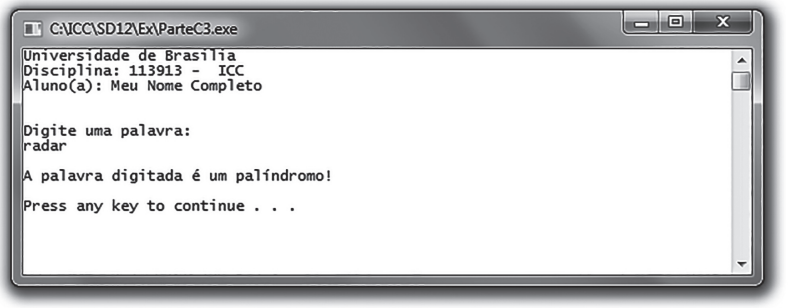


a) Frases iguais.

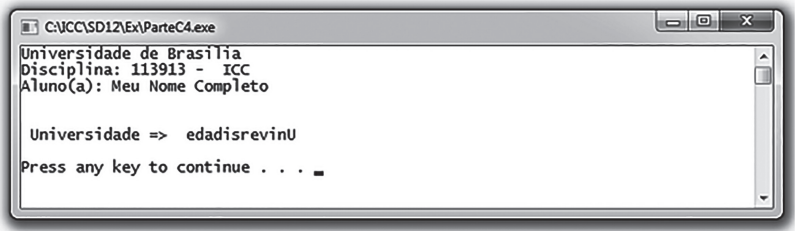


b) Frases diferentes.

3)


C:\ICC\SD12\Ex\ParteC3.exe
Universidade de Brasilia
Disciplina: 113913 - ICC
Aluno(a): Meu Nome Completo
Digite uma palavra:
radar
A palavra digitada é um palíndromo!
Press any key to continue . . .

4)


C:\ICC\SD12\Ex\ParteC4.exe
Universidade de Brasilia
Disciplina: 113913 - ICC
Aluno(a): Meu Nome Completo
Universidade => edadisrevinU
Press any key to continue . . .

Recomendações de Leitura

Leitura para iniciantes em programação:

BANAHAN, Mike; BRADY, Declan; DORAN, Mark. **The C book**. 2. ed. Boston: Addison-Wesley, 2003.

DAMAS, Luis M. Dias. **Linguagem C**. 10. ed. Rio de Janeiro: LTC, 2007.

DEITEL, Paul; DEITEL, Harvey M. **C**: How to Program. 7. ed. Nova Iorque: Pearson Education, 2013.

GRIFFITHS, David; GRIFFITHS, Dawn. **Head First C**. 6. ed. Boston: O'Reilly Media, 2012.

KOCHAN, Stephen G. **Programing in C**. 3. ed. Indianapolis: Sam's Publishing, 2005.

MIZRAHI, Victorine Viviane. **Treinamento em linguagem C**. 2. ed. São Paulo: Pearson Prentice Hall, 2008.

OUALLINE, Steve. **Practical C Programming**. 3. ed. Boston: O'Reilly media, 1997.

PRATA, Stephen. **C Primer Plus**. 6. ed. Boston: Addison-Wesley, 2014.

Leitura para pessoas com conhecimento intermediário em programação:

FEUER, Alan R. **The C Puzzle Book**. 1. ed. Boston: Addison-Wesley, 1998.

HANSON, David R. **C Interfaces and Implementations**. 3. ed. Boston: Addison-Wesley, 3rd Edition, 2001.

KLEMENS, Ben. **21st Century C**. 2. ed. Boston: O'Reilly Media, 2014.

PLAUGER, P. J. **The Standard C Library**. 1 ed. Nova Iorque: Prentice Hall, 1992.

REEK, Kenneth. **Pointers on C**. 1. ed. Nova Iorque: Pearson, 1997.

SCHILDT, Herbert. **C completo e total**. 3. ed. São Paulo: Makron Books, 1997.

SCHREINER, Axel-Tobias. **Object-oriented Programming with ANSI-C**. 1999. Disponível em: <https://www.cs.rit.edu/~ats/books/ooc.pdf>. Acesso em: 5 set. 2016.

SEDGEWICK, Robert. **Algorithms in C**. 3. ed. Boston: Addison-Wesley, 1997.

Leitura para pessoas com conhecimento intermediário ou avançado em programação:

LINDEN, Peter van der. **Expert C Programming**: deep c secrets. 1 ed. Nova Iorque: Prentice Hall, 1994.

PERRY, John W. **Advanced C Programming by Example**. 1. ed. [s.L.]: Pws Pub Co, 1998.

Bibliografia

BANAHAN, Mike; BRADY, Declan; DORAN, Mark. **The C book**. 2. ed. Boston: Addison-Wesley, 2003.

DAMAS, Luis M. Dias. **Linguagem C**. 10. ed. Rio de Janeiro: LTC, 2007.

DEITEL, Paul; DEITEL, Harvey M. **C**: How to Program. 7. ed. Nova Iorque: Pearson Education, 2013.

FEUER, Alan R. **The C Puzzle Book**. 1. ed. Boston: Addison-Wesley, 1998.

GRIFFITHS, David; GRIFFITHS, Dawn. **Head First C**. 6. ed. Boston: O'Reilly Media, 2012.

HANSON, David R. **C Interfaces and Implementations**. 3. ed. Boston: Addison-Wesley, 3rd Edition, 2001.

KLEMENS, Ben. **21st Century C**. 2. ed. Boston: O'Reilly Media, 2014.

KOCHAN, Stephen G. **Programing in C**. 3. ed. Indianapolis: Sam's Publishing, 2005.

LINDEN, Peter van der. **Expert C Programming**: deep c secrets. 1 ed. Nova Iorque: Prentice Hall, 1994.

MIZRAHI, Victorine Viviane. **Treinamento em linguagem C**. 2. ed. São Paulo: Pearson Prentice Hall, 2008.

OUALLINE, Steve. **Practical C Programming**. 3. ed. Boston: O'Reilly media, 1997.

PERRY, John W. **Advanced C Programming by Example**. 1. ed. [s.L.]: Pws Pub Co, 1998.

PLAUGER, P. J. **The Standard C Library**. 1 ed. Nova Iorque: Prentice Hall, 1992.

PRATA, Stephen. **C Primer Plus**. 6. ed. Boston: Addison-Wesley, 2014.

REEK, Kenneth. **Pointers on C**. 1. ed. Nova Iorque: Pearson, 1997.

SCHILDT, Herbert. **C completo e total**. 3. ed. São Paulo: Makron Books, 1997.

SCHREINER, Axel-Tobias. **Object-oriented Programming with ANSI-C**. 1999. Disponível em: <https://www.cs.rit.edu/~ats/books/ooc.pdf>. Acesso em: 5 set. 2016.

SEDGEWICK, Robert. **Algorithms in C**. 3. ed. Boston: Addison-Wesley, 1997.

Este roteiro de experimentos serve de auxílio para aulas de laboratório em linguagem C de cursos de graduação na área de exatas e para pessoas que desejam aprender e praticar os conceitos dessa linguagem.

Cuidadosamente elaborado para pessoas sem qualquer conhecimento de linguagem de programação C, o conteúdo é aqui descrito com simplicidade e centrado na realização de 12 experimentos, que servem de apoio aos exercícios básicos e de fixação, todos com soluções desenvolvidas.

Cada capítulo apresenta uma breve descrição da teoria necessária para realizar os exercícios em um fluxo lógico, possibilitando que o discente compreenda de maneira eficaz e rápida os exercícios práticos do ANSI C.

Esta obra foi concebida na experiência docente dos autores com as aulas práticas de programação de uma disciplina dos cursos de Engenharias da Faculdade UnB Gama da Universidade de Brasília e faz parte da coleção "Introdução à Linguagem C". Completa essa coleção o livro “Linguagem C – Aprendendo com Exercícios Resolvidos”.

ISBN: 978-85-79883-10-1
(Coleção Completa)

ISBN: 978-85-79883-07-1