1ª Edição

Linguagem C

Ricardo Antonello

www.antonello.com.br

ISBN: 978-65-00-01892-9

O autor

Ricardo Antonello é mestre em Ciência da Computação pela Universidade Federal de Santa Catarina – UFSC e bacharel em Ciência da Computação pelo Centro Universitário de Brasília – UniCEUB. Em 2000 iniciou sua carreira no Banco do Brasil e atuou no BB Private Banking, desde 2006 é professor universitário. Na Universidade do Oeste de Santa Catarina - Unoesc foi coordenador do Núcleo de Inovação Tecnológica – NIT e da Pré-Incubadora de Empresas. Atualmente é professor no Instituto Federal Catarinense – IFC e atua em projetos na área de visão computacional. É um entusiasta das atividades do Polo de Inovação Vale do Rio do Peixe – Inovale.

Contato: ricardo@antonello.com.br

Blog: www.antonello.com.br

Foto da Capa

A foto da capa deste livro foi orgulhosamente realizada por mim em minha viagem a São Francisco, uma das cidades do Vale do Silício no estado da Califórnia nos EUA. Veja a foto completa abaixo:

Prefácio

O mundo está mudando a uma velocidade incrível e para os incautos que acham que as coisas vão desacelerar, lamento, não vão! É por isso que cada vez mais a programação se tornará importante não só na vida de qualquer profissional de engenharia como na de qualquer cidadão.

Steve Jobs, fundador da Apple, Next, Pixar, criador do iPhone e outras várias tecnologias disruptivas pensa o seguinte sobre o aprendizado de programação: “everyone should learn how to program, because it teaches you how to think” que em uma tradução deste autor significa “Todos devem aprender a programar, porque a programação ensina a pensar”.

Os conteúdos nesta obra são os mesmos encontrados em muitos livros sobre programação em C, contudo, deixo minha contribuição com uma abordagem que considero mais didática. Mas não se restrinja apenas a este livro, pois você pode e deve abraçar outros conteúdos e materiais, principalmente sobre uma área que você goste de “programar”.

Aproveito aqui para agradecer ao colega e professor Ricardo Kerschbaumer que me indicou o software Flowgorithm utilizado para gerar os fluxogramas deste livro. Vale a pena baixar e usar, principalmente para aqueles que são iniciantes na arte da programação.

Boa sorte e não esqueça: Viver em sintonia com o mundo é muito mais interessante! Fique ligado sempre! Prosperidade para todos nós! Ricardo Antonello

Sumário

1. Sistema binário

Para garantir que este livro seja como a maioria dos livros de programação, ou seja, confusos, vamos iniciar com a seguinte frase: “A programação começou antes dela mesma”.

Entendeu? Bem, eu explico: Qualquer coisa processada por um computador, celular, tablet ou qualquer dispositivo digital opera com a aritmética binária, ou seja, processa as informações através de números, especificamente dos números zero e um. Esse jeito de calcular as coisas surgiu antes do computador e foi criado por *Boole*[1], mas outras mentes brilhantes da época também colaboraram para chegar onde estamos atualmente. Vou tentar resumir bastante.

Em 1642 *Pascal*[2] já havia produzido uma máquina mecânica (sem uso de energia elétrica) que realizava cálculos, contudo, as funções não podiam ser programadas. A máquina podia apenas calcular aquilo para o qual foi projetada. Foi só em 1801 que um costureiro (veja só um costureiro, quem diria) é que criou a primeira máquina programável. Na verdade, era um tear para produzir tecidos, mas com o uso de cartões perfurados inseridos nesse tear, ele podia produzir tecidos diferentes de forma automática.

Essa ideia do tear foi utilizada por vários projetos a partir de então. Dentre esses projetos, dois são muito importantes, afinal o criador deles, *Charles Babbage*[3], hoje é considerado uns dos criadores da computação. A máquina de diferenças, ainda totalmente mecânica, podia calcular funções trigonométricas e logarítmicas. Pena que a máquina só pôde ser produzida anos depois, pois o projeto de *Babbage* não tinha como ser produzido quando foi criado devido à limitação tecnológica da época. Afinal estamos falando de 1822.

Babbage também projetou uma máquina chamada de Engenho Analítico (Máquina Analítica). Ela aproveitava todos os conceitos do Tear Programável, como o uso dos cartões. Além disso, instruções e comandos também poderiam ser informados pelos cartões, fazendo uso de registradores (memórias primitivas). A precisão chegava a 50 casas decimais. Incrível pra época! Mas de novo a máquina não foi produzida por falta de tecnologia, tudo ficou apenas na fase de projeto, no papel. Contudo, esses conceitos de *Babbage* influenciam até hoje o jeito que os computadores processam informações.

Um pouco depois, em 1847, o matemático *George Boole* criou um jeito “maluco”

---

[1] George Boole (1815-1864) foi matemático e filósofo britânico, criador da álgebra booleana, fundamental para o desenvolvimento da computação moderna.
[2] Blaise Pascal (1623-1662) foi físico, matemático, filósofo e teólogo francês.
[3] Charles Babbage (1791-1871) foi cientista, matemático, filósofo, engenheiro mecânico e inventor inglês que criou o conceito do computador programável.

de representar informações e calcular operações aritméticas como soma, multiplicação e divisão, através apenas dos números zero e um. Enquanto nossa aritmética comum usa algarismos de 0 a 9, a lógica binária de *Boole* usava apenas dois, ou seja, o algarismo zero e o algarismo 1. Essa lógica é binária porque são dois valores, sempre 0 ou 1.

Apenas 100 anos depois os computadores foram criados e até hoje os computadores só entendem 0 e 1. Sendo que o zero significa desligado, falso ou "não ativo". E o número um significa ligado, ativo, ou verdadeiro. Nesse ponto você deve estar se perguntando: "O que é que eu tenho a ver com isso?" e eu lhe digo: "Muita coisa", dê um pouco de crédito a lógica que hoje faz funcionar o mundo! Preste bem atenção abaixo.

Podemos fazer contas com apenas números 0 e 1, e podemos representar com eles qualquer outro número do sistema decimal que é o que utilizamos normalmente (com dígitos de 0 a 9) da seguinte forma:

0 → 0
1 → 1
2 → 10
3 → 11
4 → 100
5 → 101
6 → 110
7 → 111
8 → 1000
9 → 1001
10 → 1010
11 → 1011
12 → 1100
13 → 1101
14 → 1110
15 → 1111
16 → 10000 e assim por diante…

Figura 1 Filme "The Imitation Game" que no Brasil foi chamado de "O Jogo da Imitação" conta a história de Alan Turing, o pai da computação. Você deve assistir esse filme!

Essa é a famosa base binária ou base 2, afinal são dois valores (0 ou 1). Explicar como fazer contas com a base binária esta fora do escopo, ou seja, da abrangência deste livro. Mas você encontrará muito material na internet pesquisando por "Lógica binária" ou "Aritmética binária". Além disso, em um próximo capítulo mostraremos como converter números decimais para binários e vice-versa. Continuando a história... foi só em 1931, que *Vannevar Bush*[4] implementou um computador com uma arquitetura binária propriamente dita, usando os bits 0 e 1. A base decimal (de 0 a 9) exigia que a eletricidade assumisse 10 tensões diferentes (por exemplo 1 volt, 2 volts, 3 volts, etc...),

[4] Vannevar Bush (1890-1974) foi engenheiro, inventor e político estadunidense, conhecido pelo seu papel político no desenvolvimento da bomba atômica e pela ideia do memex, visto como um conceito pioneiro, precursor da world wide web.

o que era muito difícil de ser controlado. Por isso, *Bush* fez uso da lógica de Boole, onde somente dois níveis de voltagem já eram suficientes e… deu certo!

E aí veio a segunda guerra mundial, e com ela muito incentivo ao desenvolvimento dos computadores. A máquina era usada para decifrar mensagens dos inimigos e realizar outros cálculos com fins bélicos. Nesse período o que mais se destacou foi o *Mark I*, no ano de 1944, criado pela Universidade de Harvard (EUA), e o *Colossus*, em 1946, criado por *Allan Turing*[5]. Em 2015 foi lançado um filme muito bom sobre Alan Turing chamado "O Jogo da Imitação". O título original do filme é "The Imitation Game", vale a pena assistir.

Alan Turing definiu muitas coisas que hoje são utilizadas no seu *smartphone*, então agradeça a ele todo dia quando acordar e usar o WhatsApp antes de escovar os dentes.

Outro computador que se destacou muito foi o *ENIAC (Electrical Numerical Integrator and Calculator)*, desenvolvido pelos cientistas norte-americanos John Eckert e John Mauchly. Esta máquina era em torno de mil vezes mais rápida que qualquer outra que existia na época.

O *ENIAC* só usava válvulas eletrônicas, possuindo dimensões enormes (do tamanho de um container), utilizava quilômetros de fios, chegando a atingir temperaturas muito elevadas, o que frequentemente causava problemas de funcionamento. Dessa época bem o termo "bug" de computador, porque quando um bug (inseto em inglês) entrava dentro dessas máquinas gigantescas, isso causava um curto circuito que desligava a máquina. Até hoje quando um computador apresenta algum erro o sinônimo para isso é "bug".

A segunda geração de computadores eletrônicos melhorou muito por dois motivos. As válvulas que eram grandes e esquentavam muito foram substituídas por transistores menores e mais "frios". Além disso, surgiu o circuito impresso em pastilhas de cilício, permitindo que vários transistores estivessem no mesmo chip. O IBM 7030 foi o primeiro dessa geração e podia ser comprado por empresas, apesar de custar 13 milhões de dólares. Foi o primeiro computador comercial.

A partir disso vieram os minicomputadores (terceira geração) e depois os microcomputadores (quarta geração) que estão aí até hoje.

Em 1975 foi lançado o Altair 8800 com o processador 8080 da Intel. Foi para esse computador que Bill Gates adaptou o primeiro sistema operacional da Microsoft, o Basic. Depois veio o MS-DOS e só depois o Windows. Ainda na época do Altair, Steve Jobs e Steve Wozniak criaram o "Apple 1", primeiro computador da Apple. E o resto? Bem... o resto você já sabe.

---

[5] Alan Mathison Turing (1912-1954) foi matemático, lógico, criptoanalista e cientista da computação britânico. Muitos o consideram o pai da ciência da computação devido à formalização do conceito de algoritmo e computação com a máquina de Turing. Ele também é pioneiro na inteligência artificial.

1.1. CONVERSÕES DE BASE

Antes de começar a programar é preciso entender como o computador representa a informação. E quando falamos em informação estamos falando basicamente de números, pois os caracteres, incluindo letras e outros símbolos são representados internamente no computador por números. Explicaremos isso com calma nos capítulos seguintes. Agora iremos ficar na representação dos números dentro do computador.

No capítulo anterior falamos que *Boole* criou a álgebra booleana que pode representar qualquer número apenas com os símbolos 0 (zero) e 1 (um). Para compreender melhor este conceito vamos partir do que já conhecemos que são os números na base decimal.

A base decimal utiliza 10 símbolos para representar os números. Esses símbolos são os números 0 (zero) até 9 (nove) já que para representar o número dez precisamos de dois símbolos, o número 1 (um) seguido de um 0 (zero), ou seja, 10.

Para iniciar esse processo vamos decompor os números decimais de um jeito que provavelmente você nunca viu! ☺

Sabemos que no número 123 (cento e vinte e três) é composto pelo número mais a direita (3) que é a unidade. Somamos a unidade a dezena que é o número 2 e que, por ser dezena, o multiplicamos por 10 resultando em 20. Aí na sequência temos o número 1 que é a centena, ou seja, é o número 1 multiplicado por 100 que é igual a 100. Por fim, somando a centena mais a dezena mais a unidade temos: 100 + 20 + 3 = 123.

Podemos ainda decompor esse cálculo da seguinte maneira:

$$1 * 100 + 2 * 10 + 3 * 1 = 123$$

Perceba que a unidade é multiplicada por 1, a dezena por 10, a centena por 100 e assim por diante. Essa sequência de 1, 10, 100 pode ser decomposta na base decimal da seguinte forma: $1 = 10^0, 10 = 10^1$ *e* $100 = 10^2$. Ou seja, 1, 10 e 100 são $10^0, 10^1$ *e* 10^2 respectivamente. Aí está a base 10. Entendendo os números decimais dessa forma fica fácil entender a mudança para a base binária, pois o que iremos mudar é só a base que passa de 10 para 2. Veja a fórmula genérica abaixo:

$$a * 10^n + [...] + b * 10^2 + c * 10^1 + d * 10^0$$

Ou ainda:

$$a * base^n + [...] + b * base^2 + c * base^1 + d * base^0$$

Onde a onde base é igual a 10 para a base decimal.

1.2. BASE BINÁRIA

Para convertermos números na base binária para a base decimal o que precisamos fazer é utilizar os conceitos acima e substituir a base por 2 (base binária).

Dessa forma o número na base binária 101 é decomposto em $1*2^2 + 0*2^1 + 1*2^0$ conforme abaixo:

$$1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 1 * 4 + 0 * 2 + 1 * 1 = 5$$

Já o número 10 na base binária é o número 2 na base decimal, da seguinte forma:

$$1 * 2^1 + 0 * 2^0 = 1 * 2 + 0 * 1 = 2$$

Por fim, o número 111 na vase binária é 7 na base decimal, pois:

$$1 * 2^2 + 1 * 2^1 + 1 * 2^0 = 1 * 4 + 1 * 2 + 1 * 1 = 7$$

Para complementar o entendimento veja a tabela abaixo:

Tabela 1: Conversão da base binária para decimal.

Número na base binária	Conversão para base decimal	Número na base decimal
0 0 0 0 =	0 * **8** + 0 * **4** + 0 * **2** + 0 * **1** =	0
0 0 0 1 =	0 * **8** + 0 * **4** + 0 * **2** + 1 * **1** =	1
0 0 1 0 =	0 * **8** + 0 * **4** + 1 * **2** + 0 * **1** =	2
0 0 1 1 =	0 * **8** + 0 * **4** + 1 * **2** + 1 * **1** =	3
0 1 0 0 =	0 * **8** + 1 * **4** + 0 * **2** + 0 * **1** =	4
0 1 0 1 =	0 * **8** + 1 * **4** + 0 * **2** + 1 * **1** =	5
0 1 1 0 =	0 * **8** + 1 * **4** + 1 * **2** + 0 * **1** =	6
0 1 1 1 =	0 * **8** + 1 * **4** + 1 * **2** + 1 * **1** =	7
1 0 0 0 =	1 * **8** + 0 * **4** + 0 * **2** + 0 * **1** =	8
1 0 0 1 =	1 * **8** + 0 * **4** + 0 * **2** + 1 * **1** =	9
1 0 1 0 =	1 * **8** + 0 * **4** + 1 * **2** + 0 * **1** =	10
1 0 1 1 =	1 * **8** + 0 * **4** + 1 * **2** + 1 * **1** =	11
1 1 0 0 =	1 * **8** + 1 * **4** + 0 * **2** + 0 * **1** =	12
1 1 0 1 =	1 * **8** + 1 * **4** + 0 * **2** + 1 * **1** =	13
1 1 1 0 =	1 * **8** + 1 * **4** + 1 * **2** + 0 * **1** =	14
1 1 1 1 =	1 * **8** + 1 * **4** + 1 * **2** + 1 * **1** =	15

Portanto, para converter números binários para a base decimal precisamos multiplica-los pela base, já para converter os números decimais para a base binária teremos que usar a divisão.

Para converter o número 123 na base decimal para a base binária usamos o seguinte procedimento: Dividimos (com divisão inteira) sucessivamente o número 123 por 2 e vamos acumulando os restos. Ao final invertemos a ordem dos restos acumulados e teremos o número equivalente a 123 na base binária. Difícil de entender não é mesmo? Pois bem, vamos desenhar ;)

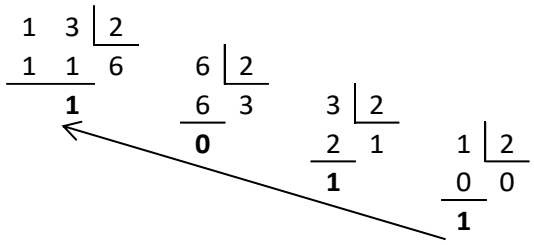


Figura 2: Conversão de número na base decimal para base binária. Número 13 na base decimal é igual a 1101 na base binária.

Já para converter o número 123 na base decimal para a base binária temos:

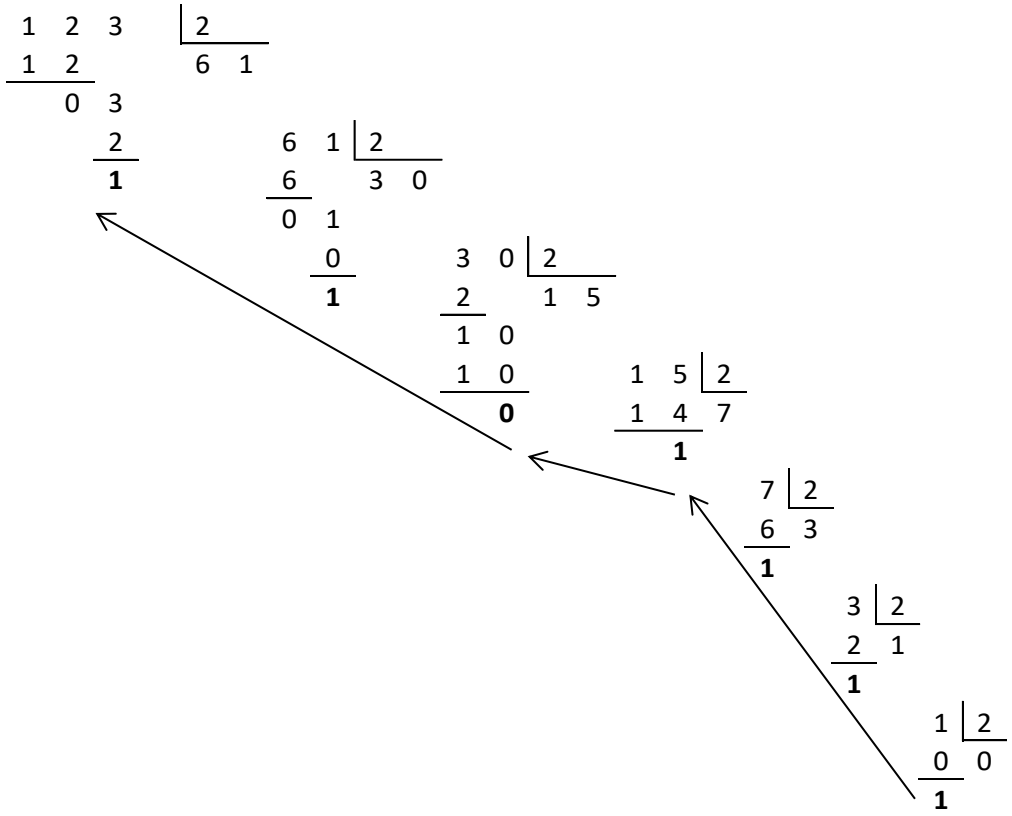


Figura 3: Conversão de número na base decimal para a base binária onde temos 123 = 1111011.

Partindo da figura acima, basta pegar a sequência dos restos das divisões de baixo para cima que teremos o número na base binária correspondente, ou seja, 1111011 é igual a 123.

É importante lembrar que em algumas linguagens de programação o número binário vem precedido de ‘0b’ então 1111011 em binário é expresso por 0b1111011.

1.3. BASE HEXADECIMAL

O mesmo procedimento utilizado para converter números binários para decimais e vice-versa é utilizado para converter números na base hexadecimal para decimal e vice-versa. Número em hexadecimal utilizam a base 16, ou seja, cada digito pode ter 16

valores. Dessa forma, para os números de 0 a 9 o sistema é igual ao decimal de 10 a 15 são utilizadas letras conforme abaixo:

Decimal		Hexadecimal
0	=	0
1	=	1
...	=	...
9	=	9
10	=	A
11	=	B
12	=	C
13	=	D
14	=	E
15	=	F

Figura 4: De 0 (zero) a F (quinze) termos 16 valores possíveis para cada dígito na base hexadecimal.

Portanto, o número 8 em decimal é igual ao número 8 na base hexadecimal. Mas o número 11 em decimal não é igual a 11 em hexadecimal já que 11 em hexadecimal é igual a 17 em decimal, veja:

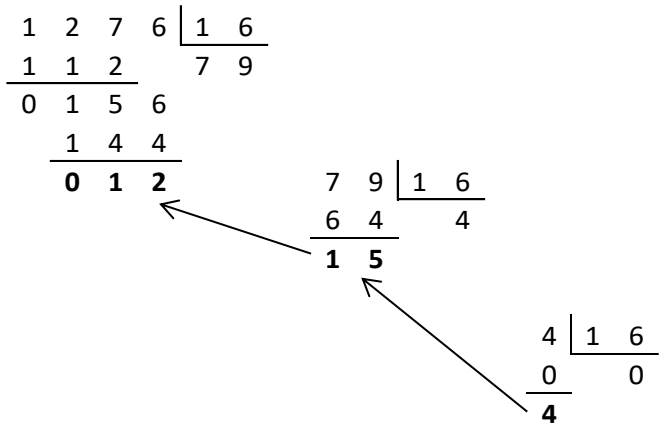
$$1 * 16^1 + 1 * 16^0 = 1 * 16 + 1 * 1 = 17$$

É importante lembrarmos que o número 11 em hexadecimal também pode ser expresso por 0x11. Então 0x11 = 17, ou seja, 11 na base hexadecimal é igual a 17 na base decimal.

Já o número 0x4FC que é igual a 1276 pode ser convertido da seguinte forma:

$$4 * 16^2 + F * 16^1 + C * 16^0 = 4 * 256 + 15 * 16 + 12 * 1 = 1276$$

Para fazer o inverso, ou seja, converter o número 1276 em decimal para a base hexadecimal precisamos seguir o mesmo procedimento da conversão binária, mas ao invés de dividir o número por 2 iremos dividir por 16. Dessa forma, os restos de cada divisão poderão ser de 0 a 15 que está dentro da base de representação hexadecimal. Veja:



Dessa forma, os restos das divisões são 4, 15 e 12 onde 4 permanece igual, 15 é transformado em F e 12 é transformado em C. No fim temos 4FC ou 0x4FC que é a solução correta.

1.4. EXERCÍCIOS PROPOSTOS

1.1) Qual é a representação binária para o número 7 em decimal?

1.2) Qual é a representação binária para o número 16 em decimal?

1.3) Qual é a representação binária para o número 34 em decimal?

1.4) Qual é a representação binária para o número 66 em decimal?

1.5) Qual é a representação binária para o número 127 em decimal?

1.6) Qual é a representação binária para o número 222 em decimal?

1.7) Qual é a representação binária para o número 3000 em decimal?

1.8) Qual é a representação binária para o número 5555 em decimal?

1.9) Qual é a representação binária para o número 19765 em decimal?

1.10) Qual é a representação hexadecimal para o número 7 em decimal?

1.11) Qual é a representação hexadecimal para o número 16 em decimal?

1.12) Qual é a representação hexadecimal para o número 23 em decimal?

1.13) Qual é a representação hexadecimal para o número 66 em decimal?

1.14) Qual é a representação hexadecimal para o número 127 em decimal?

1.15) Qual é a representação hexadecimal para o número 222 em decimal?

1.16) Qual é a representação hexadecimal para o número 3000 em decimal?

1.17) Qual é a representação hexadecimal para o número 5555 em decimal?

1.18) Qual é a representação hexadecimal para o número 19765 em decimal?

1.19) Qual é a representação decimal para o número 0x00001?

1.20) Qual é a representação decimal para o número 0x11?

1.21) Qual é a representação decimal para o número 0x3DF?

1.22) Qual é a representação decimal para o número 0x350?

1.23) Qual é a representação decimal para o número 0x3D45A?

1.24) Qual é a representação decimal para o número 0xAAA?

1.25) Qual é a representação binária para o número 0x00001?

1.26) Qual é a representação binária para o número 0x11?

1.27) Qual é a representação binária para o número 0x3DF?

1.28) Qual é a representação binária para o número 0x350?

1.29) Qual é a representação binária para o número 0x3D45A?

1.30) Qual é a representação binária para o número 0xAAA?

1.31) Qual é a representação decimal para o número 0b111?

1.32) Qual é a representação decimal para o número 0b001?

1.33) Qual é a representação decimal para o número 0b1001001?

1.34) Qual é a representação decimal para o número 0b001001001?

1.35) Qual é a representação decimal para o número 0b11001?

1.36) Qual é a representação decimal para o número 0b001111001?

1.37) Qual é a representação decimal para o número 0b1010101010?

1.38) Qual é a representação hexadecimal para o número 0b111?

1.39) Qual é a representação hexadecimal para o número 0b001?

1.40) Qual é a representação hexadecimal para o número 0b1001001?

1.41) Qual é a representação hexadecimal para o número 0b001001001?

1.42) Qual é a representação hexadecimal para o número 0b11001?

1.43) Pesquise sobre como funciona a base octal. Opcionalmente, realize as conversões dos números dos exercícios acima “de” e “para” a base octal.

2. Algoritmos e fluxogramas

Normalmente, quando um aluno primeiro aprende a programar, ele geralmente usa uma das linguagens de programação baseadas em texto. Dependendo do idioma, isso pode ser fácil ou extremamente difícil.

Muitas linguagens exigem que você escreva linhas de códigos confusas apenas para exibir o texto "Alô, mundo!" na tela. Usando fluxogramas, você pode se concentrar em conceitos de programação, em vez de em todas as complicações de uma linguagem de programação típica.

Como este livro é destinado a quem ainda não sabe programar, este capítulo vai mostrar o que é um algoritmo e como utilizar fluxogramas para criar algoritmos antes de partirmos diretamente para a Linguagem C.

Para isso usaremos um excelente software chamado Flowgorithm[6]. Seu nome é uma associação das palavras fluxograma e algoritmo e é exatamente do que precisamos agora.

A ideia aqui é utilizar fluxogramas simples para montar os algoritmos e entender comandos de decisão e laços de repetição por exemplo. Depois de entender a lógica de programação, é fácil aprender um dos principais idiomas.

Também é possível executar programas diretamente no Flowgorithm já que ele pode converter interativamente o fluxograma em mais de 18 idiomas que incluem: C#, C ++, Java, JavaScript, Lua, Perl, Python, Ruby, Swift, Visual Basic .NET e VBA (usados no Office).

2.1. ALGORITMOS

Antes de mostrarmos o primeiro fluxograma vamos definir o que são algoritmos para entender como os fluxogramas podem nos ajudar. Um algoritmo é uma sequência finita de instruções não ambíguas, em resumo são passos para resolver uma tarefa. O conceito de algoritmo existe há séculos e o uso do conceito pode ser atribuído a matemáticos gregos, por exemplo a Peneira de Eratóstenes e o algoritmo de Euclides.

O conceito de algoritmo é frequentemente ilustrado pelo exemplo de uma receita de bolo 😋, embora normalmente os algoritmos úteis sejam mais complexos. Algoritmos podem repetir passos (iterações) ou tomar decisões (se x maior que y então...).

Veja um exemplo de algoritmo para fazer um bolo:

1. Aqueça o forno a 180 ºC

---

[6] Disponível em: <http://www.flowgorithm.org>. Acesso em: 18 abr. 2018.

2. Unte uma forma redonda
3. Bata 50g de manteiga junto com 200g de açúcar até ficar cremoso
4. Junte 3 ovos, um a um
5. Junte 80g de chocolate derretido
6. Adicione aos poucos 220g de farinha peneirada
7. Deite a massa na forma
8. Leve ao forno durante 30 minutos

Portanto, o algoritmo pode ser uma lista simples de frases que indicam tarefas específicas em uma determinada ordem.

Perceba que os algoritmos são expressos em linguagem natural, portanto, a vantagem é a simplicidade e facilidade de compreensão. Contudo, sua principal desvantagem é que a linguagem natural é bem diferente da linguagem utilizada pelos computadores. Por exemplo, em linguagem natural, seja português ou inglês, uma palavra pode ter vários significados dependendo do contexto no qual são utilizadas. Isso não pode acontecer em uma linguagem de programação pois o computador não saberia que decisão tomar[7].

Usar uma lista de frases pode ficar pouco produtivo quando temos muitas decisões ou repetições a fazer. Por exemplo veja o algoritmo para calcular a média de 3 notas de um aluno e informar se ele passou com média maior ou igual a 7 ou se ele pegou exame. Perceba que caso o aluno tenha ficado em exame será importante calcular a média final juntamente com a nota recebido do exame.

1. Calcule a média simples das 3 notas
2. Se a média das notas é maior ou igual a 7 então informe que o aluno passou sem exame.
3. Se a média das notas é menor que 7 então informe que o aluno pegou exame.
4. Se o aluno pegou exame receba a nota do exame.
5. Calcule a média final somando a média das notas mais a nota do exame e dividindo tudo por 2.
6. Se a média final é maior ou igual a 5 o aluno passou com exame.
7. Senão o aluno reprovou.

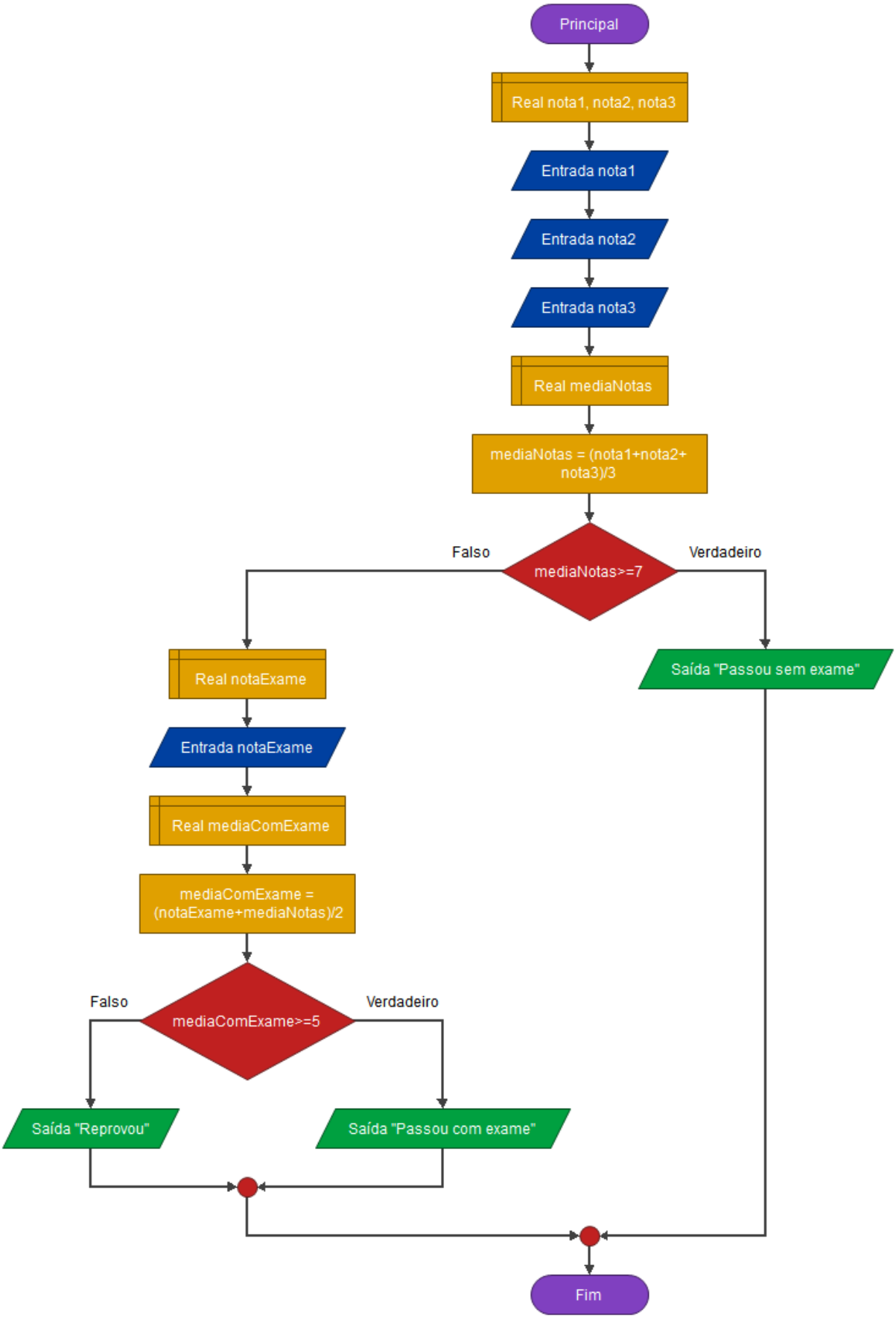
Perceba que ainda é possível entender o que está ocorrendo, mas certamente seria mais fácil visualizar o processo em um fluxograma, afinal, uma imagem vale mais que mil palavras.

2.2. FLUXOGRAMAS

Vamos iniciar mostrando o fluxograma do algoritmo do cálculo de notas.

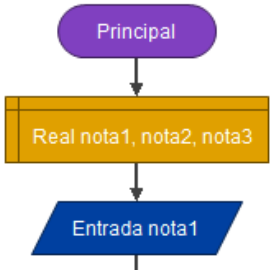
---

[7] Não estou levando em conta nesta afirmação as tecnologias de inteligência artificial ou redes neurais, pois, neste caso, já é possível compreender contextos onde palavras de qualquer língua estão inseridas. Com inteligência artificial também é possível capacitar o programa a tomar decisões em condições de incertezas.


Principal
Real nota1, nota2, nota3
Entrada nota1
Entrada nota2
Entrada nota3
Real mediaNotas
mediaNotas = (nota1+nota2+ nota3)/3
Falso
mediaNotas>=7
Verdadeiro
Real notaExame
Saída "Passou sem exame"
Entrada notaExame
Real mediaComExame
mediaComExame = (notaExame+mediaNotas)/2
Falso
mediaComExame>=5
Verdadeiro
Saída "Reprovou"
Saída "Passou com exame"
Fim

Perceba como a informação utilizando fluxogramas fica muito mais estruturada. Tenho certeza que assim que você começar a utilizar fluxogramas nos seus algoritmos vai gostar dessa fantástica ferramenta. Com o tempo não será mais necessário colocar o passo a passo no fluxograma, pois isso será automática para programas mais simples. Contudo, se você está iniciando na programação faça SEMPRE o fluxograma dos seus códigos.

Para descrever melhor o fluxograma vamos detalhar alguns componentes. Primeiro, perceba que antes da entrada de dados, simbolizada pelo paralelogramo azul, temos um retângulo amarelo com a frase “Real nota1, nota2, nota3”. Esse componente está declarando, ou seja, criando as variáveis sob as quais iremos trabalhar. Neste caso as 3 variáveis (nota1, nota2 e nota3) irão receber valores do tipo Real que são números reais. Outros tipos são números inteiros (Int) e letras (Char). Após definir os nomes é possível fazer a ‘entrada’ de cada nota. Ao executar o programa você verá que a entrada é o momento que o código para e espera você informar algum número pelo teclado.

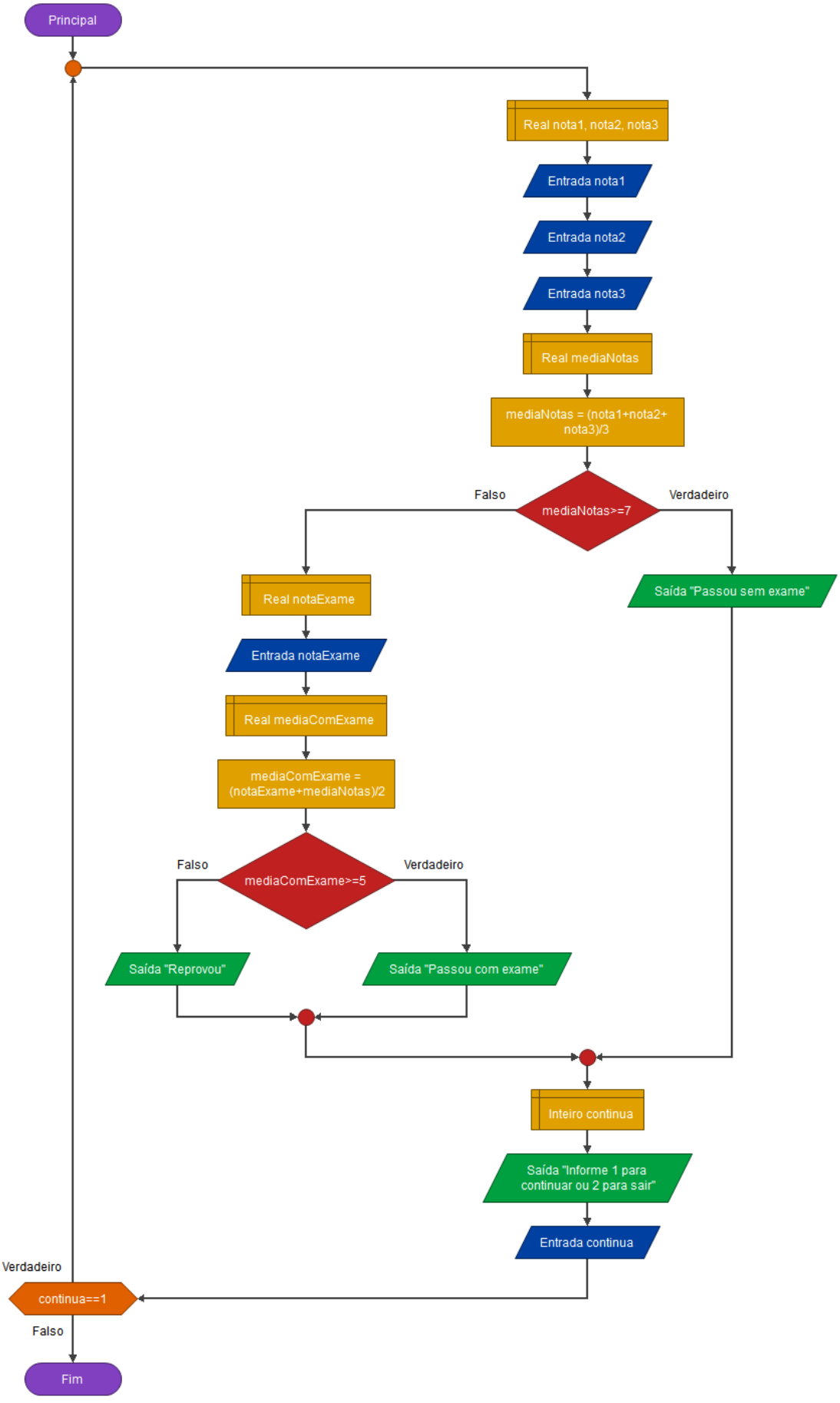


O fluxograma não possui nenhum laço de repetição, mas, podemos incluir caso seja necessário refazer os cálculos de notas mais de uma vez. Veja a alteração no algoritmo:

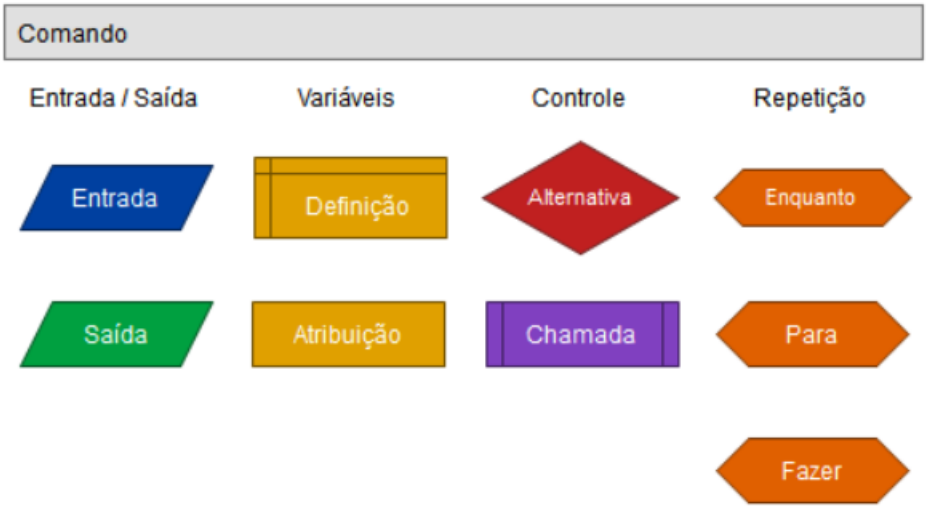
1. Receber as 3 notas
2. Calcule a média simples das 3 notas
3. Se a média das notas é maior ou igual a 7 então informe que o aluno passou sem exame.
4. Se a média das notas é menor que 7 então informe que o aluno pegou exame.
5. Se o aluno pegou exame receba a nota do exame.
6. Calcule a média final somando a média das notas mais a nota do exame e dividindo tudo por 2.
7. Se a média final é maior ou igual a 5 o aluno passou com exame.
8. Senão o aluno reprovou.
9. O usuário escolhe 1 para refazer o cálculo ou 2 para sair
10. Se o usuário escolheu 1 ir para o passo 1

Veja que incluímos um passo 1 antes inexistente, pois agora é necessário descrever que as notas serão informadas pelo usuário. Os passos 9 e 10 também foram incluídos para realizar a lógica de controle do processo.

Veja como fica o fluxograma neste caso:

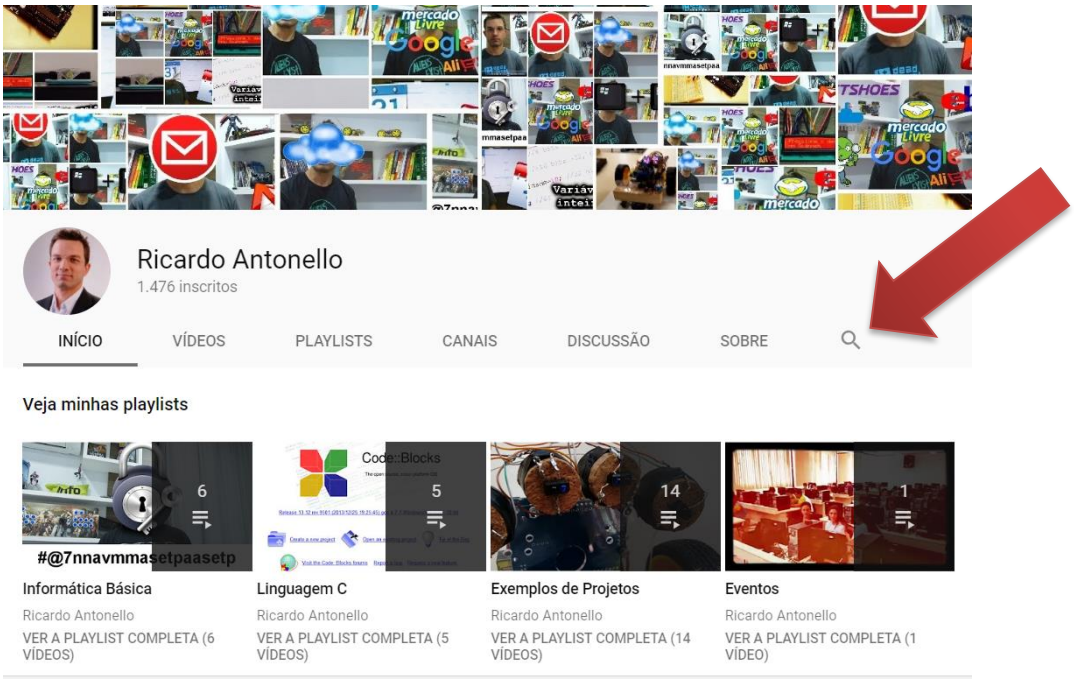

Principal
Real nota1, nota2, nota3
Entrada nota1
Entrada nota2
Entrada nota3
Real mediaNotas
mediaNotas = (nota1+nota2+ nota3)/3
Falso
mediaNotas>=7
Verdadeiro
Real notaExame
Saída "Passou sem exame"
Entrada notaExame
Real mediaComExame
mediaComExame = (notaExame+mediaNotas)/2
Falso
mediaComExame>=5
Verdadeiro
Saída "Reprovou"
Saída "Passou com exame"
Inteiro continua
Saída "Informe 1 para continuar ou 2 para sair"
Entrada continua
Verdadeiro
continua==1
Falso
Fim

Abaixo temos uma descrição de cada componente disponível. Utilizando o software Flowgorithm fica tudo muito mais fácil pois ao clicar sobre qualquer seta de fluxo a janela abaixo é exibida e basta selecionar o componente desejado.



Explore cada um dos componentes e nunca esqueça de definir uma variável antes de utilizá-la.

Se preferir acesso o meu canal no Youtube[8] para ver a construção de fluxogramas no Flowgorithm. Basta pesquisa vídeos com a palavra chave “flowgorithm”.



[8] Disponível em: <www.youtube.com/ricardoantonello>. Acesso em 22 de mar. 2020.

2.3. EXERCÍCIOS PROPOSTOS

2.1) O que é um algoritmo?

2.2) Qual é a notação dos fluxogramas para início e fim do fluxo? Desenhe abaixo:

2.3) Qual é a notação dos fluxogramas para entrada e saída? Desenhe abaixo:

2.4) Qual é a notação dos fluxogramas para comandos de decisão (controle/alternativa)?

2.5) Qual é a notação dos fluxogramas para atribuição?

2.6) Crie um algoritmo (em texto) do jogo “pedra papel tesoura”, receba as jogadas dos dois jogadores e exiba o resultado.

2.7) Faça um fluxograma para o algoritmo do exercício anterior.

2.8) Faça um algoritmo (em texto) para calcular o imposto de renda sobre aplicações em renda fixa. Para a caderneta de poupança a tributação é zero, ou seja, o investidor é isento de pagar imposto de renda. Caso a aplicação seja em títulos do tesouro, CDB ou fundos, as regras são: 22.5% para resgates em até 180 dias, 20% para resgates em até 360 dias, 17,5% para resgates em até 720 e 15% para resgates acima de 720 dias. No algoritmo, receba como entrada o tipo de aplicação, o valor aplicado e o prazo para então calcular e exibir o valor em reais do imposto devido.

2.9) Faça um fluxograma para o algoritmo do exercício anterior.

2.10) Faça um algoritmo (em texto) para calcular a nota necessária no exame, sendo que a entrada é a média. A fórmula para calcular a nota necessária no exame é:

$$Nota\ Exame = 50 - (6 * Média)/4$$

2.11) Faça um fluxograma para o algoritmo do exercício anterior.

2.12) Faça um algoritmo (em texto) que recebe três números, determina qual é o maior dentre eles e apresenta este número.

2.13) Faça um fluxograma para o algoritmo do exercício anterior.

3. Introdução à Linguagem C

3.1. HISTÓRICO

A linguagem C foi criada por Dennis Ritchie em 1972 nos laboratórios Bell para ser incluída com o sistema operacional Unix do computador PDP-11. A origem da linguagem C foi a linguagem ALGOL 60, definida em 1960. ALGOL era uma linguagem de alto nível, ou seja, permitia que o programador criasse um código genérico para rodar em qualquer máquina com um compilador compatível, sem se preocupar, por exemplo, com os endereços físicos da memória e dos registradores do processador onde o programa iria rodar. Até então, só existiam linguagem de “baixo nível” que exigiam que o programador conhecesse a fundo o hardware onde o programa iria rodar.

Os aspectos inovadores da linguagem ALGOL não foram suficientes para seu sucesso. Sete anos depois surgiu CPL (Combined Programming Language) nas universidades de Londres e Cambridge, que tinha como ideia base os aspectos da linguagem ALGOL, contudo, as dificuldades de implementação também impediram a CPL de ser amplamente aceita. O projeto da CPL foi simplificado por Martin Richards, em Cambridge que criou o Basic CPL (BCPL), uma simplificação do CPL. Já em 1970, Ken Thompson, chefe da equipe que projetou o UNIX para o PDP11 do Bell Labs, implementou um compilador BCPL batizando seu feito como linguagem de B. A linguagem B, portanto, já possuía independência da máquina onde iria ser executada, além de recursos avançados para a época como recursão. Mais tarde a evolução desse código foi feita por Dennis Ritchie.

A linguagem B não possuía tipos de dados definidos, e tinha limitações para trabalhar com caracteres. A linguagem B era muito boa com números, mas os computadores modernos da época, notadamente o PDP-11, precisavam trabalhar com conjuntos de caracteres e B não dava total suporte a isso. Então em 1971 Dennis Ritchie fez alterações na linguagem B fazendo alterações em seu compilador para produzir código compatível com tipos de dados em variáveis. Durante 1971 e 1972 a linguagem B sofreu grandes evoluções, passando a chamar "New B" (NB) e posteriormente linguagem C. Portanto, foi o projetista Dennis Ritchie, do Bell Labs, que projetou a nova linguagem, sucessora do B, que foi chamada de C.

A linguagem C foi a primeira linguagem de alto nível de propósito geral que dava ao programador condições possibilidade de desenvolver programas em qualquer área seja científica ou comercial. Em 1985, o Instituto ANSI (American National Standards Institute) estabeleceu um padrão oficial de C o chamado "C ANSI".

Em resumo, a linguagem C é uma linguagem de propósito geral, e, portanto, se adapta a qualquer tipo de projeto. É altamente portável, ou seja, pode ser compilada e executada em vários computadores diferentes. Também é extremamente rápida em tempo de execução, ou seja, na hora de executar os programas. Por fim, é importante notar que a linguagem C++ é a mesma linguagem C com o suporte adicional de orientação a objetos.

3.2. BIBLIOTECAS

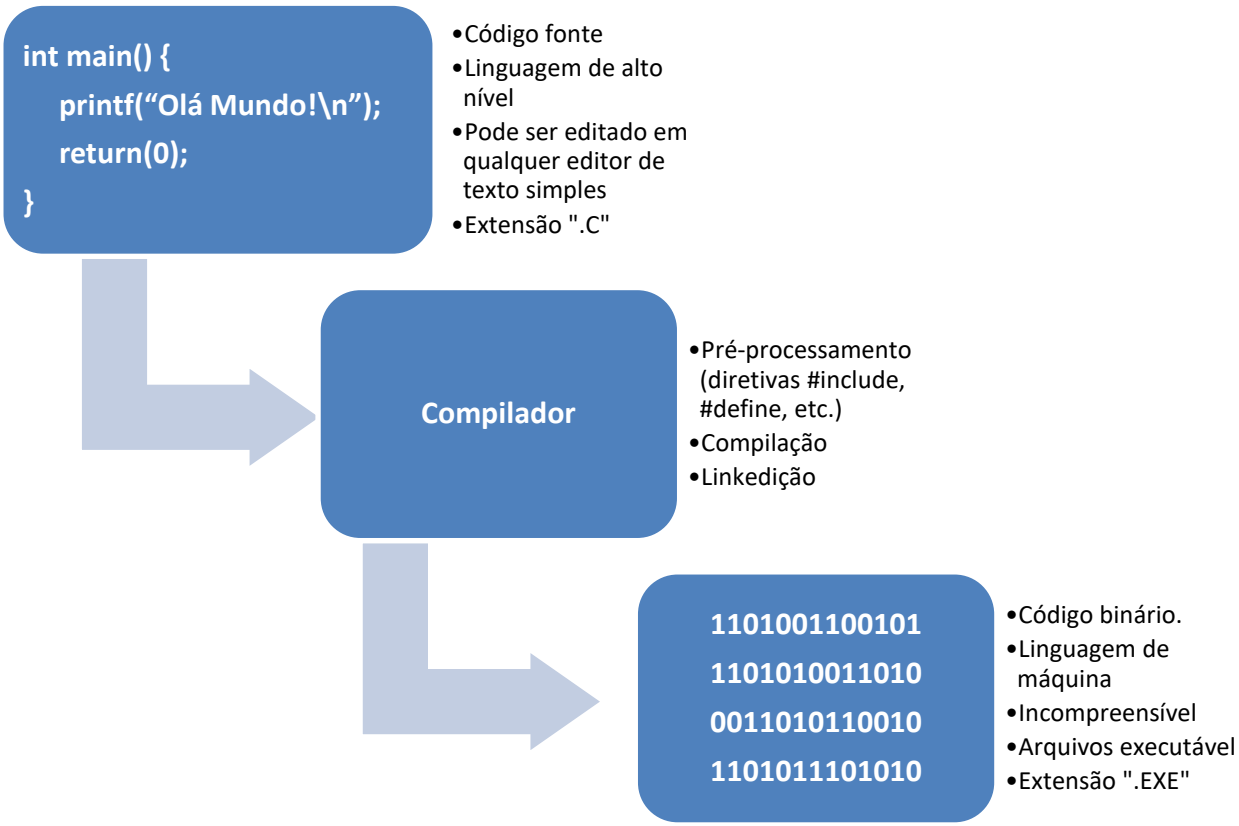
No contexto da programação de computadores, bibliotecas são subprogramas de computador, ou seja, módulos com funções prontas para serem utilizadas por outros programas. Elas podem ser anexadas ao programa em tempo de compilação ou em tempo de execução. Para isso existem dois tipos de biblioteca: a biblioteca dinâmica (dynamic library) e a biblioteca estática (static library).

As bibliotecas dinâmicas são utilizadas apenas quando o programa é executado. Neste caso os arquivos da biblioteca precisam estar disponíveis em todas as vezes que o programa é executado. Como exemplo temos os arquivos com extensão “.dll” no Windows e os arquivos com extensão “.so” no Linux.

Já as bibliotecas estáticas são incluídas no arquivo final gerado pelo processo de compilação. Neste caso, o “linker” faz a “linkedição”, ou seja, junta o código fonte já compilado escrito por você com o as bibliotecas estáticas necessárias. Na linguagem C utilizamos a diretiva “#include” para incluir uma biblioteca estática, teremos um exemplo mais adiante ainda neste capítulo.

3.3. COMO FUNCIONA O COMPILADOR

Neste início de sua caminhada no mundo da programação em Linguagem C é importante definir alguns conceitos básicos sobre código fonte, compilador e código executável.



O código fonte deve ser escrito pelo programador, é no código fonte que estão os comandos, para desenvolver um programa de computador, portanto, o código fonte deve estar elaborado em uma sequência lógica de execução onde as instruções ou comandos, em sequência, resolvam o problema em questão.

Para facilitar o desenvolvimento do código fonte, as linguagens de programação atribuem nomes compreensíveis a suas instruções, dessa forma o código fonte é legível e compreensível e pode ser compartilhado entre vários programadores ou equipes de programação.

Após a finalização do código fonte para o programa ser executado é preciso realizar o processo de compilação que irá gerar o arquivo executável que já estará em “linguagem de máquina”. A linguagem de máquina é totalmente incompreensível ao programador. Não é possível abrir um arquivo em linguagem de máquina e alterá-lo manualmente. Para isso, é preciso voltar ao código fonte que gerou o arquivo executável, fazer as alterações no código fonte e então “recompilar” o programa.

Atualmente, existem software chamados de IDE (Integrated Development Environment), ou ambiente de desenvolvimento integrado que juntam editor de texto para trabalhar no código fonte com ferramentas de compilação e execução do código gerado. Aplicativos como esses são fundamentais para facilitar e agilizar o trabalho do programador. Recomendo, para o trabalho neste livro, a IDE CodeBlocks disponível em www.codeblocks.org para Windows e Linux. Ou ainda o sistema on-line repl.it disponível em http://repl.it que pode ser usado para várias linguagens.

3.4. DEPURAÇÃO DE CÓDIGO

A função de depuração de código, em inglês chamada de “debug” ou “debugging” é um processo importantíssimo tanto para programadores profissionais como para os iniciantes. A depuração permite executar passo a passo cada linha de código de um programa, permitindo encontrar erros mais facilmente e analisando o comportamento de cada linha na execução final do software.

Existem basicamente dois tipos de erros em programação de computadores, erros em tempo de compilação e em tempo de execução. O primeiro deles e os mais fáceis de encontrar são os erros “em tempo de compilação”, ou seja, são erros que impedem que o código seja compilado, impedem que o arquivo final do programa seja gerado. Normalmente esses erros são identificados pelo compilador na linha em que ocorrem e sabendo a linha fica mais fácil para o programador encontrar o erro.

Um exemplo típico de erro em tempo de compilação na Linguagem C é errar o nome de um comando ou tentar utilizar uma variável que ainda não foi criada.

O outro tipo de erro é mais difícil de encontrar, trata-se do erro em tempo de execução. Esse erro normalmente é um erro de lógico no algoritmo programado. Este erro não é encontrado pelo compilador que irá gerar o arquivo executável normalmente. Contudo, a execução do programa não irá ocorrer conforme o esperado, gerando resultados errados ou “travando” o programa quando ele estiver sendo executado.

Para erros em tempo de execução a função de depuração de código é fundamental. Ela permitirá executar o programa passo a passo, linha a linha, e identificar a fonte do erro no código fonte que gera o comportamento inesperado na execução do programa. Procure essa função na IDE que está trabalhando, pois fazer “debug” é fundamental.

Para os programadores iniciantes é recomendável fazer a depuração do código, mesmo que não sejam encontrados erros, pois isso permite a visualização de como o código executa e facilita o entendimento e o aprendizado de programação.

3.5. MEU PRIMEIRO PROGRAMA EM C

Agora que já conhecemos um pouco sobre o processo de compilação, vamos compilar nosso primeiro programa em C. Caso ainda não tenha instalado uma IDE para trabalhar com seus programas na Linguagem C, procure por minha série de vídeos sobre Linguagem C no meu canal do Youtube. Basta procurar por “Ricardo Antonello” no Youtube para achar meu canal. Na série sobre Linguagem C o primeiro vídeo mostra como baixar e instalar o Code::Blocks IDE e o segundo vídeo mostra como criar o primeiro projeto e executar o código conforme abaixo. Outra opção, é procurar no Google por “compiladores linguagem c online”, e nesta busca utilizar algum dos sites que fornecem este serviço, o que eu mais utilizo atualmente é o Repl.it que você pode acessar através do endereço http://repl.it e criar um usuário e senhas gratuitos. Ao entrar com seu usuário e senha seus códigos ficam salvos na nuvem então você poderá editar seus programas no smartphone ou em qualquer computador sem se preocupar em fazer backups ou ter que copiar arquivos.

Veja abaixo a estrutura básica de um programa em C, este é o famoso “Olá Mundo!”, ou seja, é o menor programa que é possível fazer em Linguagem C cuja função é apenas exibir na saída padrão (normalmente a tela) a mensagem “Olá Mundo!”.

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

No código acima, a diretiva “#include” serve para incluir bibliotecas estáticas em nosso programa. A biblioteca “stdio.h” possui funções básicas para escrita em tela e leitura do teclado. Outra biblioteca muito utilizada é a “math.h” que possui funções matemáticas como raiz quadrada, potência, seno e cosseno, dentre outras.

O código da segunda linha “`int main() {`” cria a função principal de nosso programa e deve estar presente em qualquer programa que você faça na linguagem C. Já a terceira linha possui a chamada a uma função que exibe na tela a frase “Olá Mundo!”.

Por fim, temos a linha com o comando “`return(0);`” que também é fundamental para encerrar qualquer programa em C. Veremos os detalhes sobre o “return” e sobre

funções em um capítulo adiante. A última linha com um “fecha chaves” é fundamental, pois as chaves foram abertas na linha que contém a função “main()” e precisam ser fechadas. As chaves definem um bloco de código que quando aberto precisa ser fechado senão ocorre erro de compilação.

A partir de agora quando você for executar algum programa, sempre utilize o seguinte “esqueleto de código”, pois esta é a estrutura básica para iniciar um programa em C.

```
#include <stdio.h>
int main() {
  <SEU CÓDIGO VAI AQUI!>
  return(0);
}
```

Conforme nosso estudo for avançando você irá entender o que cada linha significa e o porquê da necessidade delas. Por enquanto apenas lembre-se de incluir seu código onde está descrito abaixo e “dentro” da função “main()”.

3.6. COMENTÁRIOS

Este é o momento oportuno para falar de comentários na Linguagem C. Comentários são textos que ficam dentro do código fonte, mas são ignorados pelo compilador, ou seja, são textos que só o programador pode ver e que não alteram o programa. Eles existem para permitir comentários sobre linhas de código que as vezes não são simples de entender, principalmente para os iniciantes.

É possível criar uma linha de comentário em C utilizando duas barras em sequência assim: “//”. Tudo que estiver a partir do “//” até o final da linha será um comentário.

Caso você queira um comentário de várias linhas então o melhor é iniciar o comentário com “/*” e finalizar com “*/” pois dessa forma tudo (inclusive várias linhas) que estiverem entre esses comandos serão comentários.

Veja um exemplo com o código do “Olá Mundo” que vimos anteriormente.

```
#include <stdio.h> //biblioteca
#include <stdlib.h> //biblioteca
int main() {
  //Isso é um comentário de apenas uma linha
  printf("Olá Mundo!\n");
  /* Isso
  Também é um
  Comentário. Mas pode ter várias linhas... */
  return(0);
}
```

Este código continua executando sem problemas e os comentários que estão em

negrito no código acima serão ignorados pelo compilador.

3.7. EXERCÍCIOS PROPOSTOS

3.1) Qual é a função do compilador na criação de programas em C?

3.2) Por que é necessário utilizarmos uma linguagem de programação? Não seria possível programar diretamente em linguagem de máquina?

3.5) Faça uma pesquisa na internet sobre quais funções estão disponíveis na biblioteca “math.h”.

3.6) Faça uma pesquisa na internet sobre quais funções estão disponíveis na biblioteca “stdio.h”.

3.7) O que é a deputação de código? Para que serve a depuração?

3.3) Para desenvolver um programa de computador é necessário elaborar uma sequência lógica de instruções de modo que estas instruções resolvam o problema em questão. O conjunto de instruções que podem ser utilizadas para resolver os problemas é definido pela linguagem de programação utilizada. Para facilitar a tarefa dos programadores as linguagens de programação atribuem nomes compreensíveis a suas instruções, e estas instruções são então convertidas para a linguagem do computador. Sobre esse processo é correto afirmar:

a) O compilador tem a função de converter linguagem de máquina em código de alto nível.

b) Instruções em linguagem de alto nível são compreensíveis aos seres humanos.

c) Um programa em linguagem de máquina é reconhecido apenas por computadores com sistema Windows.

d) Programas em linguagens de programação como a Linguagem C não precisam ser compilados antes de serem executados.

e) Nenhuma das alternativas está correta.

4. Variáveis, operadores e funções de entrada e saída

Em quase toda a linguagem de programação é possível acessar diretamente a memória, contudo, na maioria das vezes isso não é uma vantagem. O acesso direto à memória significa que você deverá gerenciar a posição numérica (célula de memória 1, célula de memória 2, célula de memória 53.427...) para cada informação que utilizar, e isso é muito complexo e sujeito a erros.

Por isso é que em C e em outras linguagens você pode dar nomes para espaços na memória. Esses espaços em memória são utilizados para armazenar valores, ou seja, você pode incluir o que quiser lá dentro e mudar essas informações no decorrer da execução do seu programa. Dessa forma, esses “espaços em memória” são chamados de variáveis.

Na linguagem C existem basicamente três tipos de dados: números inteiros, números reais e caracteres. Para cada tipo de dados existem variáveis que podem assumir diferentes tamanhos para armazenar as informações conforme veremos a seguir.

Tipos básicos de dados na Linguagem C:

- char: Caractere: O valor armazenado é um caractere. Caracteres geralmente são armazenados em códigos (usualmente o código ASCII[9]).
- int: Número inteiro é o tipo padrão e o tamanho do conjunto que pode ser representado normalmente depende da máquina em que o programa está rodando.
- float: Número em ponto flutuante de precisão simples. São conhecidos normalmente como números reais.
- void: Este tipo serve para indicar que um resultado não tem um tipo definido. Uma das aplicações deste tipo em C é criar um tipo vazio que pode posteriormente ser modificado para um dos tipos anteriores.

4.1. TIPOS NUMÉRICOS BÁSICOS

Para armazenar números inteiros, temos o tipo int. Para criar uma variável deste tipo utilizamos o seguinte formato:

```
<tipo> <nome da variável>;
```

Como exemplo para criar a variável chamada “Idade” para armazenar a idade de

[9] ASCII (American Standard Code for Information Interchange ou Código Padrão Americano para o Intercâmbio de Informação)

alguém temos:

```
int idade;
```

Importante notar que a Linguagem C é “case sensitive” ou seja, sensível a capitulação da letra, então “int” é diferente de “Int” que é diferente de “INT”. Os comandos são todos em letras minúsculas.

Após criada a variável “idade” conforme comando acima podemos atribuir um valor para ela da seguinte maneira.

```
idade = 18;
```

Perceba que utilizamos o caractere “=” para fazer isso. Em Linguagem C o caractere “=” é utilizado para atribuição de valores em variáveis, então ao encontrar um caractere “=” lemos “atribui” ou “recebe”. Dessa forma o comando acima deve ser interpretado como “idade recebe 18”.

Outras variações são possíveis para estes comandos de criação e atribuição de variáveis, veja:

```
int idade, idade1, idade2, temperatura;
```

Acima temos a criação de quatro variáveis inteiras no mesmo comando, separadas apenas por vírgula. Perceba que o “ponto e vírgula” ocorre apenas no final. Nenhuma variável recebeu atribuição de valor. Podemos atribuir valor em todas ou apenas em algumas conforme abaixo:

```
int idade=12, idade1, idade2=35, temperatura;
```

Veja que apenas algumas variáveis receberam valores, isso é perfeitamente possível. Contudo, não esqueça de atribuir algum valor a uma variável antes de utilizar o valor que esta dentro dela. A linguagem C não inicializa, ou seja, não atribui automaticamente valores “zero” para variáveis, então, ao ler um valor de uma variável criada mas não “inicializada”, ou seja, que ainda não recebeu nenhuma valor, o que ocorre é que a área da memória RAM do computador que esta vinculada a variável poderá conter um valor antigo, de outro processamento, ou seja, um “lixo” da memória que trará resultados imprevisíveis para o seu programa.

Para reforçar o aprendizado do que vimos até agora, vamos desenvolver alguns programas utilizando o que vimos acima e o operador “+” que é utilizado para somar valores. Veja:

```
int a=7, b=4, soma;
soma = a + b;
printf(“%i”, soma);
```

A função “printf” acima será explicada no capítulo seguinte, não se preocupe com isso agora. Ao executar o código acima não esqueça de incluir o “esqueleto básico” de

um programa em C conforme vimos no capítulo anterior. O código final será

```
#include <stdio.h>
int main() {
  int a=7, b=4, soma;
  soma = a + b;
  printf("%i", soma);
  return(0);
}
```

Se tudo ocorrem bem, o resultado em tela deverá ser o número 11. Também vale a pena notar que a linha contendo “soma = a + b” não tem o “int” na frente da variável “soma” pois essa variável já foi criada na linha acima. Caso não fosse incluída a “soma” na linha de cima, aí sim era necessário ter o “int” conforme: “int soma = a + b;”.

Você deve ter percebido que a variável temperatura citada acima foi criada com o tipo de dados inteiro. Isso não é uma boa ideia porque a temperatura normalmente requer um grau de precisão de pelo menos uma ou duas casas decimais. Utilizando uma variável inteira a temperatura terá que pular de 22 graus célsius para 21 ou 23 graus célsius. Neste caso o melhor é utilizar uma variável que armazene números reais, no caso, do tipo “float”. Veja:

```
float temperatura1 = 17.3;
float temperatura2 = 22.56;
```

Podemos realizar a mesma tarefa feita pelos dois comandos acima em uma única linha:

```
float temperatura1 = 17.3, temperatura2 = 22.56;
```

Neste caso perceba que os números possuem a parte fracionária que pode possuir precisão de até 6 casas decimais para variáveis do tipo “float”. Para mais precisão é necessário utilizar outro tipo de dados que veremos adiante.

4.2. NOMES DE VARIÁVEIS

É importante conhecer as regras dos nomes de variáveis. Em C os nomes de variáveis seguem os seguintes critérios:

- São aceitos somente letras, números e o caractere “_”.
- Deve começar com letras ou com o caractere “_”.
- Podem possuir números desde que não como primeiro caractere.
- Não podem ter espaços.

Exemplos: idade, idade1, idade_1, _idade, _idade1, idade01, _idade01, melhor_idade

Uma convenção na linguagem C é que todos os nomes de variáveis devem

possuir apenas letras minúsculas pois letras maiúsculas são reservadas para nomes de “constantes” na linguagem C (que veremos adiante). Por se tratar de convenção, o código irá funcionar normalmente se você utilizar letras maiúsculas, contudo, não é recomendado por dificultar a leitura do código por outros programadores.

4.3. TIPOS BOOLEANOS

O tipo “boolean” ou “booleano é um tipo de dados que armazena apenas dois valores, verdadeira ou falso. Contudo, na linguagem C padrão ANSI não existem variáveis do tipo booleano, então consideramos valores diferentes de zero como verdadeiros (true) e iguais a zero como falso (false).

4.4. CARACTERES

Um tipo de dados importante em C é o tipo “char”. Ele é na verdade um tipo de dados numérico, com tamanho de 8 bits, ou seja, 1 byte e consequentemente com capacidade para armazenar valores inteiros de 0 a 255. Contudo, em C utilizamos a tabela de código “*ASCII - American Standard Code for Information Interchange*” que atribui um símbolo para cada número de 0 a 255. Dessa forma, a letra “A” é representada pelo número 65, a letra “B” pelo 66. Já o letra “a” em caixa baixa, ou seja, minúscula é representada pelo número 97, a letra “b” por 98 e assim por diante. Para conhecer os outros código rode o programa abaixo. Não se preocupe em entender os códigos, eles serão explicados em breve. Também não se preocupe com o “bip” que irá ouvir ao executar o código, pois um dos códigos da tabela ASCII é sonoro, então o “bip” é normal.

```
int main(){
    int i;
    for(i=0;i<256;i++)
        printf("%i=%c\n",i,i);
    return 0;
}
```

Veja que no código criamos uma variável inteira “int” e mesmo assim exibimos caracteres na tela ao executar o programa. Isso ocorre porque caracteres são representados internamente por números e são apenas exibidos como caracteres. O tipo char, portanto, é um tipo inteiro com o propósito específico de armazena caracteres.

Importante notar que cada variável “char” armazena um e somente um caractere. Para armazenar uma palavra ou frase devemos utilizar “strings” ou vetores de caracteres. Teremos adiante um capítulo específico para isso.

4.5. MODIFICADORES DOS TIPOS BÁSICOS

Modificadores podem ser aplicados aos tipos de dados básico. Estes modificadores são palavras que alteram o tamanho do conjunto de valores que o tipo pode representar. Por exemplo, um modificador permite que possam ser armazenados

números inteiros maiores. Um outro modificador obriga que só números sem sinal possam ser armazenados pela variável. A Tabela abaixo mostra todos os tipos básicos definidos no padrão ANSI.

Tipo	Tamanho em Bytes	Faixa Mínima
char	1	-128 a 127
unsigned char	1	0 a 255
signed char	1	-128 a 127
int	4	-2.147.483.648 a 2.147.483.647
unsigned int	4	0 a 4.294.967.295
signed int	4	-2.147.483.648 a 2.147.483.647
short int	2	-32.768 a 32.767
unsigned short int	2	0 a 65.535
signed short int	2	-32.768 a 32.767
long int	4	-2.147.483.648 a 2.147.483.647
signed long int	4	-2.147.483.648 a 2.147.483.647
unsigned long int	4	0 a 4.294.967.295
float	4	Seis digitos de precisão
double	8	Dez digitos de precisão
long double	10	Dez digitos de precisão

O modificador “unsigned” que significa “sem sinal” é utilizado para eliminar um bit que define se o número armazenado é positivo ou negativo. Dessa forma, a variável irá armazenar somente valores positivos e este bit é utilizado para guardar dados o que praticamente dobra a capacidade de armazenamento da variável.

Também é importante notar que ao aumentar a capacidade de armazenamento das variáveis normalmente o tamanho em bytes e a velocidade necessária para processar a variável (ler, escrever, somar) também aumenta. Caso a programação esteja sendo feita para dispositivos móveis, ou com pouca memória é importante escolher os menores tipos de dados possíveis para melhorar a performance do código.

4.6. CONSTANTES

Constantes são usadas para armazenar valores que não podem ser modificados durante a execução. Uma constante precisa ser declarada usando a diretiva de pré-processador *#define*. A definição é a seguinte:

```
#define <nome_da_constante> <valor>
```

A declaração da constante deve ser feita no início do programa logo após as diretivas de importação de bibliotecas *#include*. O padrão na linguagem C é utilizar letras maiúsculas ao declarar uma constante, pois, dessa forma, podemos facilmente diferenciá-las das variáveis que por convenção devem ser declaradas em letras minúsculas.

As constantes são acessadas e processadas muito mais rapidamente que variáveis, portanto, é recomendável utilizar constantes sempre que possível. Veja alguns

exemplos de declaração de constantes:

```
#define MAX 100
#define ISS 0.05
#define LED1 33
#define LED2 34
#define ERRO "Atenção! Erro identificado! Contate suporte."
```

4.7. FUNÇÕES DE SAÍDA

Estudaremos primeiro as funções de saída para depois entendermos como funcionam as funções de entrada, pois é com as funções de saída que conseguiremos exibir em tela os valores das variáveis que aprendemos no capítulo anterior.

A linguagem C tem uma saída padrão que já é pré-configurada como sendo o monitor do sistema. Quando falamos em saída, basicamente trabalhamos com dados sendo “impressos” ou exibidos no monitor. Contudo, alterando a saída padrão do sistema é possível imprimir os dados em impressoras, arquivos ou saídas de rede, dentre outros dispositivos.

4.7.1. Funções puts() e putchar()

As três principais funções de saída da Linguagem C são a “*printf*”, a “*puts*” e a ”*putchar*”. A primeira é uma função bem completa, permite formatar os dados como veremos a seguir. Já a segunda e a terceira são bastante simples. A função “*puts*” permite a impressão ou exibição de “*strings*” na tela. A “*putchar*” apenas a impressão de um caractere.

Neste ponto é importante definirmos o que é uma “*string*”. Uma *string* nada mais é que uma cadeia de caracteres, um conjuntos de letras que podem conter ou não números e outros símbolos, inclusive espaços. Na linguagem C uma *string* sempre deve estar entre aspas duplas. Veja o exemplo abaixo.

```
puts(“Eu sou uma string”);
putchar('c') //c é um caractere (sempre entre aspas simples)
```

Este código para compilar e executar precisa estar dentro da função *main()* portanto o código completo fica assim:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  puts(“Eu sou uma string!”);
  putchar('c');
  return(0);
}
```

Este programa irá exibir na tela a frase: Eu sou uma string! Incluindo, inclusive, o ponto de exclamação no final que faz parte da *string*. Após vem o caractere ‘c’.

A função *puts()* também inclui um caractere especial de mudança de linha ao final da *string*, ou seja, é como se fosse pressionada a tecla “Enter” após a impressão da *string* na tela. Então qualquer coisa impressa depois será exibida na linha de baixo.

4.7.2. Função printf()

A função printf() tem seu nome derivado da junção de “print” que é imprimir traduzido do inglês com a letra ‘f’ de “function” que pode ser traduzida como função. Portanto, printf é a “*print function*” ou seja “função imprimir”.

Aqui é importante destacar que no Brasil usamos muito o termo imprimir para indicar impressão em papel, aquilo que sai das impressoras. Contudo, neste contexto de programação, imprimir pode ser na tela, na rede, no arquivo ou em outros destinos.

O primeiro argumento de printf é uma string de controle, e sendo string precisa estar entre aspas duplas. Os outros argumentos serão opcionais dependendo do que estiver presente nesta primeira string. Veja abaixo os exemplos:

printf(“Isto foi impresso na tela”);

Neste caso não tivemos os argumentos opcionais. Apenas a string que é o primeiro argumento será impressa em tela.

```
printf("Idade = %d", idade);
```

Aqui temos dois argumentos, sendo o primeiro a string e o segundo a variável chamada idade. Perceba que os argumentos de uma função são separados por vírgulas dentro dos parênteses que formam a função. O exemplo acima imprime na tela a frase Idade = 22 sendo que 22 é o valor da variável idade. É importante notar que a sequência de conversar %d fará com que o valor de uma variável inteira seja apresentado na tela. Esta variável deverá ser um argumento na função printf().

No caso de termos apenas uma sequência de conversar %d então teremos apenas mais um segundo argumento que deve indicar o nome de uma variável inteira.

Como em C letras ou caracteres isolados utilizam a tabela ASCII e são considerados inteiros sem sinal de 8 bits é possível termos o seguinte:

```
int idade = ‘A’;
printf("Idade = %d", idade);
```

E teremos impresso na tela Idade = 65, o que não faz muito sentido, mas irá executar sem qualquer erro.

Perceba que com uma sequência de formatação temos um argumento a mais na função printf(). Caso tenhamos mais de uma sequência de formatação então teremos mais de um argumento. Se as variáveis inteiras a1 e a2 possuírem respectivamente os valores 34 e 109 a saída do código abaixo será: Tenho 34 anos e quero chegar aos 109.

```
printf("Tenho %d anos e quero chegar aos %d.", a1, a2);
```

Veja que a ordem é fundamental pois o primeiro %d exibirá a variável que esta no segundo argumento e o segundo %d exibirá o valor da variável do terceiro argumento.

Se v1 é uma variável do tipo caractere com valor 'X', então a execução de

```
printf("Este pode ser o %c da questao!”, v1);
```

Imprime na tela a frase “Este pode ser o X da questão!”. Então observe que cada tipo de variável tem sua própria sequência de formatação.

Até aqui você já pôde observar que a função printf() não tem um número fixo de argumentos. Ela poderá ter quantos você precisar. Contudo, use as sequências corretas de formatação que deve começar sempre com ‘%’. Para variáveis inteiras temos:

%c imprime o conteúdo da variável com representação ASCII;

%d ou %i imprime o conteúdo da variável com representação decimal com sinal;

%u imprime o conteúdo da variável com representação decimal sem sinal;

%o imprime o conteúdo da variável com representação octal sem sinal;

%x imprime o conteúdo da variável com representação hexadecimal sem sinal.

Uma largura de campo pode ser opcionalmente especificada logo após o caráter %, como em %12d para especificar que o número decimal terá reservado um espaço de doze caracteres para sua representação. Se a largura de campo for negativa, com um sinla ‘-‘ antes do número forçamos o alimento a esquerda. Para a conversão de variáveis do tipo long, use o caractere l da seguinte forma %ld.

Para variáveis ponto flutuante temos:

%f imprime o conteúdo da variável com representação com ponto decimal;

%e imprime o conteúdo da variável com representação em notação científica (exponencial);

%g formato geral, escolhe a representação mais curta entre %f e %e.

Por fim temos o %s que é usado apenas para exibição de *strings*. Veremos isso mais adiante neste livro já que *strings* em C são nada mais que vetores de caracteres, ou seja, sequências de caracteres que internamente são armazenados como vetores. Dedicaremos um capítulo para isso adiante já que é um assunto de suma importância.

Como para a representação inteira, uma largura de campo pode ser especificada para números reais. Por exemplo, %12.3f especifica que a variável será apresentada em um campo de doze caracteres com uma precisão de três dígitos após o ponto decimal.

Importante notar que na maioria das vezes usamos %i para inteiros e %f para pontos flutuantes. No %f algumas variações são permitidas. Exemplo, tendo o valor 1.23 na variável v7 temos:

```
printf("%f", v7); // imprime 1.230000
printf("%.3f", v7); // imprime 1.230
printf("%.2f", v7); // imprime 1.23
printf("%.1f", v7); // imprime 1.2
printf("%6.2f", v7); /* imprime 1.23 colocando espaços à esquerda para que
no total, o número tenha 6 caracteres, contando o ponto decimal como um
desses caracteres.  */
printf("%06.2f" , v7); /* imprime 001.23 colocando espaços à esquerda para
que no total, o número tenha 6 caracteres, contando o ponto decimal como um
desses caracteres. */
printf("%+06.2f", v7); /* imprime +01.23 colocando sinais positivos quando o
número é positivo. Isso não interfere em nada quando o número é negativo,
pois neste caso sempre aparece o sinal – antes do número, conforme seria
esperado. */
// Como exemplo final
printf("%+08.2f", v7); // imprime: +0001.23
```

Outra característica muito interessante do *printf()* é a possibilidade de utilizar caracteres de escape. Esses caracteres começam sempre com uma barra invertida, veja a tabela abaixo:

Comando	Comportamento ou caracter representado
\a	Beep (Aviso sonoro)
\b	Backspace (Apaga o caracter anterior)
\f	Formfeed (avança linha)
\n	Newline/Line Feed (Equivalente ao [Enter])
\r	Carriage Return (Volta ao início da linha)
\t	Horizontal Tab (Tabulação)
\v	Vertical Tab (Tabulação vertical)
\\	Backslash (Imprimr uma barra invertida)
\'	Single quotation mark (Imprime aspas simples)
\"	Double quotation mark (Imprime aspas duplas)

Sem dúvida o comando mais utilizado é o '\n' como no exemplo:

```
printf("Meu nome esta abaixo\nRicardo Antonello"); // imprime na tela:

Meu nome esta abaixo
Ricardo Antonello
```

4.8. FUNÇÕES DE ENTRADA

Assim como temos o puts() e o printf() temos o gets() e o scanf(). Ambos são, respectivamente, as funções básicas de entrada de dados.

4.8.1. Funções gets() e getchar()

No caso do gets() que é derivado do inglês *get* (pegar) mais ‘s’ de *string* temos uma função que dada uma variável do tipo *string* (vetor de caracteres) captura uma *string* completa incluindo os espaços. Ou seja, o término da captura só ocorre quando o usuário pressionar a tecla [Enter].

Exemplo:

```
char p1[10]; //cria variável para armazenar uma string
puts("Informe a palavra: ");
gets(p1); //captura entrada do usuário
puts(p1); //imprime o que foi capturado
```

O código acima realiza esta captura. Importante notar que o gets() captura inclusive os espaços em branco, diferente do scanf() que veremos adiante.

A função *getchar()* captura um único caractere, veja:

```
char c;
c = getchar(); //captura o caractere
```

4.8.2. Função scanf()

A função scanf é mais versátil, contudo, exige alguns cuidados. Ela trabalha com sequência de formatação como o printf(), veja:

Comando	Descrição
i	Inteiro incluindo octal (precedido por zero) e hexadecimal (0x)
d *or* u	Inteiro decimal
ld ou li	Inteiro decimal longo (long int)
o	Inteiro octal
x	Inteiro Hexadecimal
f, e, g	Ponto Flutuante
lf	Ponto Flutuante de precisão dupla (double)
c	Caracter
s	String

Portanto para capturar uma string usamos scanf(“%s, &variavel); já para capturar um inteiro scanf(“%d, &variavel); sendo que variável é o nome da variável que deve ser do mesmo tipo e o caractere ‘&’ é obrigatório pois scanf() acessa diretamente a memória

para guardar o valor capturado e o ‘&’ é um operador que informa ao scanf() o endereço em memória da variável informada.

A vantagem do scanf() além da sua versatilidade de poder capturar qualquer tipo de dados é a seguinte:

```
scanf("%d%c%d", &v1, &op, &v2);
```

Este comando captura algo como 2+3 separando o número 2 em v1, o número 3 em v2 e o caracter ‘+’ em op.

A desvantagem é a complexidade, a necessidade do ‘&’ e para string a impossibilidade de capturar strings com espaço, pois para scanf() se você informar uma string “Ricardo Antonello” apenas o nome “Ricardo” será capturado e o resto ignorado pois o espaço entre o nome e sobrenome é entendido como finalização da string.

4.9. OPERADORES

Operadores em Linguagem C são símbolos utilizados para realizar as operações aritméticas elementares.

Abaixo temos a tabela dos operadores aritméticos. Veja que além dos tradicionais soma, subtração, multiplicação e divisão temos o operador resto. Importante não confundir o ‘%’ do operador resto com o ‘%’ da máscara interna de funções como *printf* já que apesar de ser o mesmo caractere as funções são completamente diferentes.

Operador	Finalidade	Exemplo	Resultado
+	Adição	2 + 2	4
-	Subtração	5 – 2	3
*	Multiplicação	5 * 2	10
/	Divisão (Quociente)	5 / 2	2 (divisão inteira)[10]
/	Divisão (Quociente)	5 / 2.0	2.5 (divisão decimal)
%	Resto	30 % 7	2

É importante notar que, assim como na matemática tradicional, contas de multiplicação e divisão ocorrem primeiro em relação a soma e subtração. Então é possível colocar quantos parênteses você quiser para garantir que cada expressão seja calculada na ordem correta. Veja os exemplos abaixo:

[10] É importante entender que os operadores aritméticos geram resultados do mesmo tipo de dados de seus operadores. Se os dois operandos forem int o resultado é int, por isso 5/2 é 2 e não 2.5, pois o resultado é inteiro e perde a parte fracionária 0.5. Contudo, para operações float com float ou float com int o resultado sempre é float, portanto, 5/2.0 ou 5.0/2, ou ainda 5.0/2.0 temos como resultado 2.5.

```
int a = 2+2*3; //valor de a será 8
int a = (2+2)*3; //valor de a será 12.
// É possível utilizar quantos parentes quisermos como no exemplo abaixo.
int a = ((2+2)*3)+(((5+2)-3)*7); //valor de a será de 40
```

Também existem operadores unários como o '++' e '--' e outros que abreviam certas operações. Veja abaixo:

Expressão Original	Expressão equivalente
x=x+k;	x+=k;
x=x-k;	x-=k;
x=x*k;	x*=k;
x=x/k;	x/=k;
x=x+1	x+=1 ou x++ ou ++x
x=x-1	x-=1 ou x-- ou --x

Importante notar a diferença de ++x para x++. Se este comando estiver isolado em uma linha o efeito é o mesmo. Mas se o comando estiver incrementando o valor de uma variável e no mesmo comando esta variável esta sendo atribuída a outra, o comportamento é diferente, veja:

```
// Exemplo 1:
int x=1;
x++;
printf("%i", x); // o valor de x é 2

// Exemplo 2:
int x=1;
++x;
printf("%i", x); // o valor de x também é 2

// Exemplo 3:
int x=1, y;
y=++x;//o valor de x usado na atribuição é o valor novo
printf("x: %i\ny: %i", x, y); // o valor de x e y será 2

// Exemplo 4:
int x=1, y;
y=x++; //o valor de x usado na atribuição é o valor antigo
printf("x: %i\ny: %i", x, y); // o valor de x será 2 e y será 1
```

4.10. OPERADORES DE BITS

Existem operadores de bits para trabalhar especificamente com números binários, contudo, não iremos trazer uma abordagem ampla agora, deixaremos isso para o último

capítulo deste livro. A título de curiosidade eles seguem abaixo:

Expressão Original	Expressão equivalente	Descrição
x=x>>k;	x>>=k;	Deslocamento de bits à direita
x=x<<k;	x<<=k;	Deslocamento de bits à esquerda
x=x&k;	x&=k;	E para bits
x=x|k;	x|=k;	OU para bits
x=x^k;	x^=k;	OU Exclusivo para bits
x=~k	Não se aplica	NOT ou Complemento de 1

4.11. FUNÇÃO SIZEOF

A função *sizeof* é usada para retornar o tamanho em bytes ocupado na memória pela variável. Isso pode variar dependendo do tipo de sistema operacional e compilador instalado. Por exemplo o tipo de dados int pode ter 4 bytes em um sistema e 2 em outro. Veja:

```
int a;
printf("%i", sizeof(a)); // irá imprimir provavelmente 4
```

4.12. CASTING

Primeiro, é importante entender que os operadores aritméticos geram resultados do mesmo tipo de dados de seus operadores. Se os dois operandos forem int o resultado é int, por isso 5/2 é 2 e não 2.5, pois o resultado é inteiro e perde a parte fracionária 0.5. Contudo, para operações float com float ou float com int o resultado sempre é float, portanto, 5/2.0 ou 5.0/2, ou ainda 5.0/2.0 temos como resultado 2.5 em formato float.

Existe uma maneira de converter um inteiro para um float matematicamente, basta multiplicar o inteiro por 1.0 pois como 1.0 é um float na multiplicação de um inteiro e um float o resultado sempre é float. Contudo, podemos fazer isso explicitamente colocando o tipo de dados entre parênteses antes da variável. Veja:

```
int a = 5;
float b = (float)a / 2; // b receberá o valor 2.5
```

Já para converter um float para um int não é possível fazer matematicamente mas é possível utilizar o casting:

```
float a = 3.5;
int b = (int)a; // b receberá o valor 3

/* abaixo a variável b também receberá o valor 3 pois o casting é feito
automaticamente */
int b = a;
```

4.13. EXERCÍCIOS PROPOSTOS

4.1) Quais as regras básicas para a criação de nomes de identificadores na linguagem C? Considere como identificadores os nomes de variáveis por exemplo.

4.2) Quais são as duas formas de fazer um comentário nos programas em C?

4.3) Construa um programa que apresenta na tela do computador o seu nome.

4.4) Descreva o funcionamento do operador % em duas situações:

a) Como resto nas operações matemáticas.
b) Como máscara no printf ou scanf.

4.5) Assinale a alternativa correta sobre os tipos básicos de dados na Linguagem C:

I - char: Caractere - O valor armazenado é um caractere. Caracteres geralmente são armazenados em códigos (usualmente o código ASCII).

II - int: Número inteiro - É o tipo padrão e tamanho de representação vai depender da máquina em que o programa foi compilado variando normalmente de 2 a 4 bytes.

III - float: Número ponto flutuante de precisão simples - São conhecidos normalmente como números reais.

IV - void: Este tipo serve para indicar que um resultado não tem um tipo definido. Uma das aplicações deste tipo em Linguagem C é criar um tipo vazio que pode posteriormente ser modificado para um dos tipos anteriores.

a) Somente I.
b) Somente I e II.
c) Somente III.
d) Somente I, II e III.
e) I, II, III e IV correspondem a tipos básicos de dados da Linguagem C.

4.6) Dados o código abaixo que compila e executa sem gerar erros, assinale a alternativa correta:

```
int main() {
  int a=7, b=4, soma;
  soma = a + b;
  printf("%i", soma);
            return(0);
}
```

a) O resultado em tela deverá será o número 11.
b) O resultado em tela deverá será o número 10.
c) O resultado em tela deverá será o número 9.
d) O resultado em tela deverá será o número 2.
e) O resultado em tela deverá será o número 28.

4.7) São regras para criar nomes de variáveis válidos em Linguagem C:

I - São aceitos somente letras, números e o caractere “_”.

II - Deve começar com letras ou com o caractere “_”.

III - Podem possuir números desde que não como primeiro caractere.

IV - Não podem ter espaços.

a) Somente I.
b) Somente I e II.
c) Somente III.
d) Somente I, II e III.
e) I, II, III e IV.

4.8) São exemplos válidos de nomes de variáveis:

I. Idade

II - idade1

III - idade_1

IV - _idade

V - _idade1

VI - idade01

VII - _idade01

VIII - melhor_idade

a) Somente I.
b) Somente I e II e III.
c) Somente III e IV
d) Somente I, II e III e IV.
e) Todos os itens de I até VIII.

4.9) Sobre a afirmação a seguir, assinale a alternativa correta: Uma convenção na linguagem C é que todos os nomes de variáveis devem possuir apenas letras minúsculas ou ao menos iniciar com uma letra minúscula, pois letras maiúsculas são reservadas para nomes de “constantes” na linguagem C. Por se tratar de convenção, o código irá funcionar normalmente se você utilizar letras maiúsculas, contudo, não é recomendado por dificultar a leitura do código por outros programadores.

a) A afirmação é verdadeira.
b) A afirmação é falsa.

4.10) Sobre a afirmação a seguir assinale a alternativa correta: O tipo boolean ou

booleano é um tipo de dados que armazena apenas dois valores, verdadeira ou falso. Contudo, na linguagem C padrão ANSI não existem variáveis do tipo booleano, então consideramos valores diferentes de zero como verdadeiros (true) e iguais a zero como falso (false).

a) A afirmação é verdadeira.
b) A afirmação é falsa.

4.11) Sobre a afirmação abaixo assinale a alternativa correta.

Um tipo de dados importante em C é o tipo “char”. Ele é na verdade um tipo de dados numérico, com tamanho de 8 bits, ou seja, 1 byte e consequentemente com capacidade para armazenar valores inteiros de 0 a 255. Contudo, em C utilizamos a tabela de código “ASCII - American Standard Code for Information Interchange” que atribui um símbolo para cada número de 0 a 255. Dessa forma, a letra “A” é representada pelo número 65, a letra “B” pelo 66. Já o letra “a” em caixa baixa, ou seja, minúscula é representada pelo número 97, a letra “b” por 98 e assim por diante.

a) A afirmação é verdadeira.
b) A afirmação é falsa.

4.12) Sobre “range” de variáveis responda.

I - O tipo char possui 1 byte e armazena de -128 até 127.

II - Com o modificado unsigned o tipo char armazena de 0 a 255.

III - O tipo char é um tipo inteiro apesar de ser utilizado para armazenar caracteres.

IV - É possível fazer contas somando os valores inteiros de duas variáveis do tipo char.

a) Somente I.
b) Somente I e II.
c) Somente III.
d) Somente I, II e III.
e) I, II, III e IV.

4.13) Quais dos seguintes nomes de identificadores são válidos e quais são inválidos? Justifique?

Variavel ( ) Válido ( ) Inválido
nome 1 ( ) Válido ( ) Inválido
entrada2 ( ) Válido ( ) Inválido
saida_2 ( ) Válido ( ) Inválido
1contador ( ) Válido ( ) Inválido
_valor_TOTAL ( ) Válido ( ) Inválido

4.14) Sobre a função “printf()” considere as afirmações abaixo e assinale a alternativa que corresponde as afirmações corretas.

I - %c imprime o conteúdo da variável com representação ASCII;

II - %d ou %i imprime o conteúdo da variável com representação decimal com sinal;

III - %u imprime o conteúdo da variável com representação decimal sem sinal;

IV - %o imprime o conteúdo da variável com representação octal sem sinal;

V - %x imprime o conteúdo da variável com representação hexadecimal sem sinal.

a) Somente I e IV.
b) Somente I e II e V.
c) Somente III e V.
d) Somente I, II e III.
e) I, II, III e IV e V.

4.15) Sobre a função “printf()” considere as afirmações abaixo e assinale a alternativa que corresponde as afirmações corretas.

I - printf(“%f”); imprime “1.230000”

II - printf(“%.3f”); imprime “1.230”

III - printf(“%.2f”); imprime “1.23”

IV - printf(“%.1f”); imprime “1.2”

V - printf(“%6.2f”); imprime “ 1.23” colocando espaços à esquerda para que no total, o número tenha 6 caracteres, contando o ponto decimal como um desses caracteres.

Assinale a alternativa correta:

a) Somente I e IV.
b) Somente I e II e V.
c) Somente III e V.
d) Somente I, II e III.
e) I, II, III e IV e V.

4.16) Sobre a função “printf()” considere as afirmações abaixo e assinale a alternativa que corresponde as afirmações corretas.

I - printf(“%06.2f”); imprime 001.23 colocando espaços à esquerda para que no total, o número tenha 6 caracteres, contando o ponto decimal como um desses caracteres.

II - printf(“%+06.2f”); imprime +01.23 colocando sinais positivos quando o número é positivo. Isso não interfere em nada quando o número é negativo, pois neste caso sempre aparece o sinal – antes do número, conforme seria esperado.

III - printf("%+08.2f", 1.23); imprime: +0001.23

Assinale a alternativa correta:

a) Somente I.
b) Somente I e II.
c) Somente I e III.
d) Somente II e III.
e) Todas são verdadeiras.

4.17) Com relação a função *print function* “printf()” e a função *put string* “puts()”, assinale a alternativa correta.

a) A função puts() permite o uso da máscara “%i”.
b) A função puts() é exatamente igual a função printf().
c) A função puts() permite o uso da máscara “%.2f”.
d) A função puts() inclui automaticamente um caractere “\n” de “nova linha” ao final da string impressa.
e) A função printf() não permite o uso de máscaras como “%i”.

4.18) Qual é a saída do programa abaixo?

```
#include <stdio.h>
int main(void) {
     float result, a=1, b=2, c=3, d=7, e=8;
     result=(b-(c*7))/(4-(a*c*a));
     printf("%f",result);
     return 0;
}
```

a) -19.000000
b) -48.000000
c) -22.000000
d) -22.00
e) 22.000000

4.19) Qual é a saída do programa abaixo?

```
#include <stdio.h>
int main(void) {
     int i=10, i2=18;
     printf("%f", ((++i+i2++)/2.0));
     return 0;
}
```

a) 14.000000
b) 14.500000
c) 15.000000
d) 14.0
e) 14.5

4.20) Referente ao código abaixo:

```
int x=7;
printf(“%d”,++x%2);
```

a) Imprime 6.
b) Imprime 7.
c) Imprime 8.
d) Imprime 1.
e) Imprime 0.

4.21) Qual é saída do seguinte programa?

```
int main() {
   int x;
   int y;
   int z;
   x=10;
   y=-20;
   z=x-y;
   printf("Resultado 1 = %d\n",z);
   z=y+x;
   printf("Resultado 2 = %d\n",z);
   z=z/(x/5);
   printf("Resultado 3 = %d\n",z);
   return(0);
}
```

4.22) Qual será a saída no terminado referente a execução do código abaixo?

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    char a=97;
    float x = 2.3;
    int resultado = x+a;
    printf("%.2f\n", resultado);
    return 0;
}
```

a) 99.3
b) 97.3
c) 99
d) 99.30
e) Nenhuma das alternativas

4.23) Qual será a saída no terminado referente a execução do código abaixo?

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    char a=97;
    float x = 2.3;
    int resultado = x+a;
    printf("%.2f\n", resultado);
    return 0;
}
```

a) 99.3
b) 97.3
c) 99
d) 99.30
e) Nenhuma das alternativas

4.24) Qual é a saída do programa abaixo se informadas as notas: 6, 8 e 10?

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
     //Media de 3 notas
     float n1, n2, n3, media=0;
     puts("Digite a primeira nota:");
     scanf("%f",&n1);
     puts("Digite a segunda nota:");
     scanf("%f",&n2);
     puts("Digite a terceira nota:");
     scanf("%f",&n3);
     printf("Media: %f", media);
     return 0;
}
```

a) Media: 0.000000
b) Media: 8.000000
c) Media: 8.0
d) 8.0
e) 0.000000

4.25) Faça um programa que lê três números reais do teclado e apresenta na tela a média destes números.

4.26) Faça um programa para ler dois números inteiros, x e y, e imprimir o quociente e o resto da divisão inteira entre eles. Utilize variáveis inteiras.

4.27) Escreva abaixo o código de um programa que calcule a média de quilômetros feitos com cada litro de combustível. Solicite a entrada de dados com quilômetros e litros e depois exiba o cálculo.

4.28) Faça um programa para resolver a equação: x=(32−y)/(2a+b). Sendo a=10, b=2 e y=5.

4.29) Escreva um programa para ler uma temperatura em graus Celsius, calcular e escrever o valor correspondente em graus Fahrenheit.

$$°F = °C \times 1{,}8 + 32$$

4.30) Escreva um programa para ler o raio de um círculo, calcular e escrever a sua área.

$$A = \pi r^2$$

4.31) Faça um programa que pede para o operador digitar uma letra, um número inteiro e um número real. Em seguida o programa deve ler estes dados, armazená-los nos tipos de dados adequados e imprimi-los na tela.

4.32) Faça um Programa que pergunte quanto você ganha por hora e o número de horas trabalhadas no mês. Calcule e mostre o total do seu salário no referido mês, sabendo-se que são descontados 14% para o Imposto de Renda, 11% para o INSS e 1% para o sindicato. Calcule os descontos e o salário líquido, e exibe um relatório conforme a tabela abaixo:

```
+ Salário Bruto: R$ <valor>
- IR (14%): R$ <valor>
- INSS (11%): R$ <valor>
- Sindicato (1%): R$ <valor>
= Salário Líquido: R$ <valor>
```

Obs.: Salário Bruto - Descontos = Salário Líquido.

4.33) Faça um Programa que receba um valor de combustível, calcule e mostre o valor desse mesmo combustível saído da refinaria e todos os valores incluídos no preço final, com base nos seguintes percentuais de despesas e impostos (Referência: http://www.petrobras.com.br/ pt/produtos/composicao-de-precos/).

- 17% - Distribuição e Revenda
- 12% - Custo Etanol Anidro
- 28% - ICMS
- 7% - CIDE, PIS/PASEP e COFINS

A saída do programa deve obedecer ao seguinte formato:

```
+ Valor na bomba: R$ <valor>
- Distribuição e Revenda (17%): R$ <valor>
- Custo Etanol Anidro (12%): R$ <valor>
- ICMS (28%): R$ <valor>
- CIDE, PIS/PASEP e COFINS (7%): R$ <valor>
= Valor da refinaria: R$ <valor>
```

Exemplo de programa

```
Entrada:
3.00

Saída:
+ Valor na bomba: R$ 3.00
- Distribuição e Revenda (17%): R$ 0.51
- Custo Etanol Anidro (12%): R$ 0.36
- ICMS (28%): R$ 0.84
- CIDE, PIS/PASEP e COFINS (7%): R$ 0.21
= Valor da refinaria: R$ 1.08
```

Importante: esta saída corresponde ao exemplo de entrada acima e quando for verificado o programa, serão utilizados outros valores.

4.34) O cardápio de uma lanchonete é dado abaixo. Prepare um algoritmo que leia a quantidade de cada item que você consumiu e calcule o valor da conta final.

- X-salada.....R$ 5.50
- X-Bacon.....R$ 7.25
- X-Tudo.....R$ 10.30
- X-Egg.....R$ 7.00
- Cerveja.....R$ 4.50
- Refrigerante.....R$ 6.00

4.35) Faça um programa para uma loja de tintas. O programa deverá pedir o tamanho em metros quadrados da área a ser pintada. Considere que a cobertura da tinta é de 1 litro para cada 3 metros quadrados e que a tinta é vendida em latas de 18 litros, que custam R$ 80,00. Informe ao usuário a quantidades de latas de tinta a serem compradas e o preço total.

4.36) Escreva um programa que calcule a fórmula de Bhaskara. As entradas serão a, b, e c referentes a uma equação de segunda grau do tipo ax^2+bx+c e a saída serão as duas raízes r1 e r2.

4.37) Escreva um programa que recebe como entrada a quantidade de maças para o transporte e o preço de cada maça. Como resultado o programa deve informar o valor total da carga e a quantidade de caixas de maça necessárias para o transporte. Considere que uma caixa de maças possui espaço para 42 unidades.

4.38) Escreva um programa para ler as dimensões de uma cozinha retangular (comprimento, largura e altura), calcular e escrever a quantidade de caixas de azulejos para se colocar em todas as suas paredes (considere que não será descontada a área ocupada por portas e janelas). Cada caixa de azulejos possui 3 m^2.

4.39) Escreva um programa que leia um número inteiro e calcule e mostre a sua decomposição em unidade, dezena, centena e milhar. Considere que o número máximo recebido via teclado será de 9999. Exemplo: A entrada 8531 terá a saída: unidade = 1

dezena = 3 centena = 5 milhar = 8.

4.40) Escreva um programa para calcular e imprimir o número de lâmpadas necessárias para iluminar um determinado cômodo de uma residência. Dados de entrada: a potência da lâmpada utilizada (em watts), as dimensões (largura e comprimento, em metros) do cômodo. Considere que a potência necessária é de 18 watts por metro quadrado.

4.41) Uma equipe de Fórmula 1 deseja calcular o número mínimo de litros que deverá colocar no tanque de seu carro de corrida para que ele possa percorrer um determinado número de voltas até o primeiro reabastecimento. Faça um programa que leia o comprimento da pista (em metros), o número total de voltas a serem percorridas em toda a corrida, o número de reabastecimentos desejados e o consumo de combustível do carro (em Km/L). Calcular e escrever o número mínimo de litros necessários para percorrer até o primeiro reabastecimento. OBS: Considere que o número de voltas entre os reabastecimentos é o mesmo.

4.42) Faça um Programa para uma loja de tintas. O programa deverá pedir o tamanho em metros quadrados da área a ser pintada. Considere que a cobertura da tinta é de 1 litro para cada 6 metros quadrados e que a tinta é vendida em latas de 18 litros, que custam R$ 140,00 ou em galões de 3,6 litros, que custam R$ 52,00. Informe ao usuário as quantidades de tinta a serem compradas e os respectivos preços em 3 situações. Sempre arredonde os valores para cima, isto é, considere latas cheias:

- Comprar apenas latas de 18 litros;
- Comprar apenas galões de 3,6 litros;
- Misturar latas e galões.

4.43) Marque a opção que em que todos os nomes de variáveis são válidos. Os nomes estão separados por vírgulas.

a) real$, errado!, correta
b) casa, olho gordo, valor
c) terminou, int, k
d) case, x, casa
e) Nenhuma das alternativas

4.44) Faça um programa que lê do teclado um número do tipo unsigned char (valores de 0 a 255) e apresenta na tela o valor dos oito bits que compões este número. Por exemplo, de a entrada for 12 a saída deve ser 0b00001100. Considere que o usuário irá realmente entrar com um valor de 0 a 255.

5. Comandos Condicionais

5.1. EXPRESSÕES LÓGICAS

Antes de iniciar os estudos com os comandos condicionais é preciso entendermos o que é uma condição. Basicamente, uma condição é uma expressão lógica cujo resultado pode assumir apenas duas possibilidades: verdadeiro ou falso.

Os operadores que usamos em condições são:

Operador	Significado
>	Maior que
<	Menor que
>=	Maior ou igual à
<=	Menor ou igual à
==	Igual a
!=	Diferente de

Estes operadores podem ser utilizados com variáveis, números ou caracteres, já que os caracteres, internamente, também são valores numéricos dentro da tabela ASCII.

Portanto, a expressão 3>5 é falsa e a expressão 5>3 é verdadeira. Cuidado com o sinal de igual, pois 3=5 irá gerar um erro de compilação, já que um sinal de igual '=' significa atribuição de valor e não é possível atribuir o valor 5 para o numero 3. Já a expressão 3==5 é falsa e a expressão 5==5 é verdadeira.

Os parênteses assim como nas expressão aritméticas podem ser usados nas expressões lógicas. Então a expressão (3<=5) é válida assim como 3<=5. Os parênteses são fundamentais para evitar ambiguidades quando usamos outros operadores lógicos conforme abaixo:

Operador	Nome	Descrição
&&	E	Conecta duas expressões lógicas. As duas precisam ser verdadeiras para o resultado ser verdadeiro, senão o resultado é falso.
||	OU	Conecta duas expressões lógicas. As duas precisam ser falsas para o resultado ser falso, senão o resultado é verdadeiro.
!	NÃO	Inverte o resultado de uma expressão lógica. Se for verdadeira se transforma em falsa e vice-versa.

Então a expressão 3>5 || 5==5 é verdadeira e podemos escrevê-la como (3>5) || (5==5) ou como ((3>5) || (5==5)) para evitar ambiguidades.

5.2. COMANDO DE CONTROLE IF

O comando “if” é o principal comando condicional. Ele possui a estrutura a seguir:

```
SE condição ENTÃO instrução
```

E na linguagem C o comando fica assim:

```
if (expressão)
  instrução;
```

Por exemplo:

```
int idade = 21;
if (idade>18)
  printf(“Já é maior de idade”);
```

Caso seja necessário ter mais de um comando dentro do if então utilize um bloco de código que inicia com { e termina com }, veja:

```
int idade = 21;
if (idade>18){
  printf(“Já é maior de idade”);
  printf(“\nEntão pode dirigir...”);
}
```

Eu sugiro utilizar o bloco de código sempre, mesmo quando tem apenas uma linha dentro, assim é mais fácil para quem é iniciante na programação. Lembre-se da identação, ou seja, tudo que esta em um bloco de código deve ficar dois espaços mais a direita.

Outra questão importante em Linguagem C é que não existe um tipo de dados booleano que assume verdadeiro ou falso, mas números inteiros ou não possuem valor lógico e a regra é apenas essa: Zero é considerado falso e qualquer coisa diferente de zero é considerada verdadeira. Veja:

```
int a=5;
if(a=3){
  printf(“Isso será impresso na tela”);
}
```

A expressão do comando *if* acima será considerada verdadeira pois em a=3 o valor 3 será atribuído para ‘a’ e ‘a’ será avaliado como expressão lógica. Como ‘a’ tem o valor 3 que é diferente de zero então o valor lógico é verdadeiro e o printf() é executado. O correto seria *if(a==3)* que neste caso seria considerado falso e o *if* não seria executado.

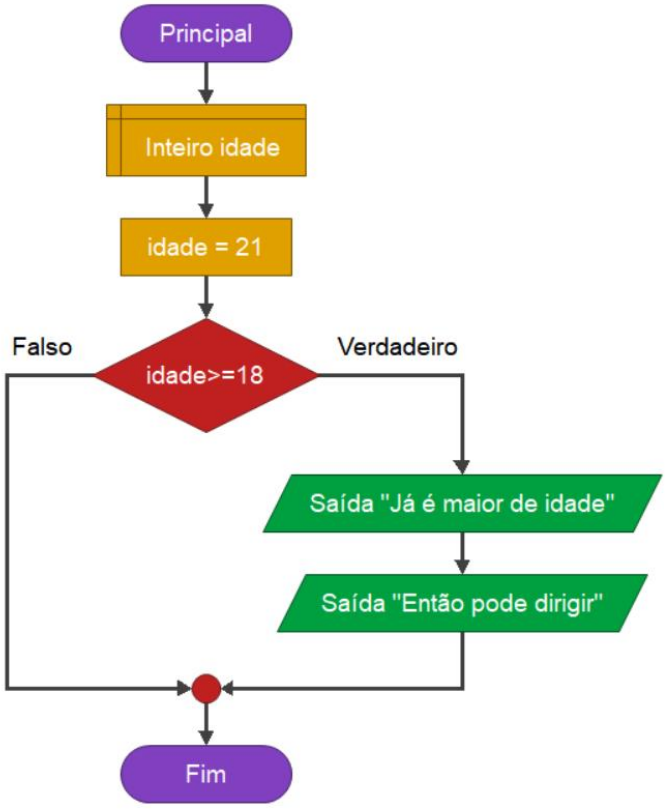
O comando *if* pode assumir várias configurações, então, nada melhor que aprendermos por exemplos. Em casa situação abaixo teremos um fluxograma

equivalente gerado no *Flowgoritm.*

5.2.1. *Decisão* simples

A decisão simples caracteriza-se por possui apenas uma expressão lógica e o resultado é verdadeiro ou falso. O *if* mais simples possível é o que vimos anteriormente, veja:

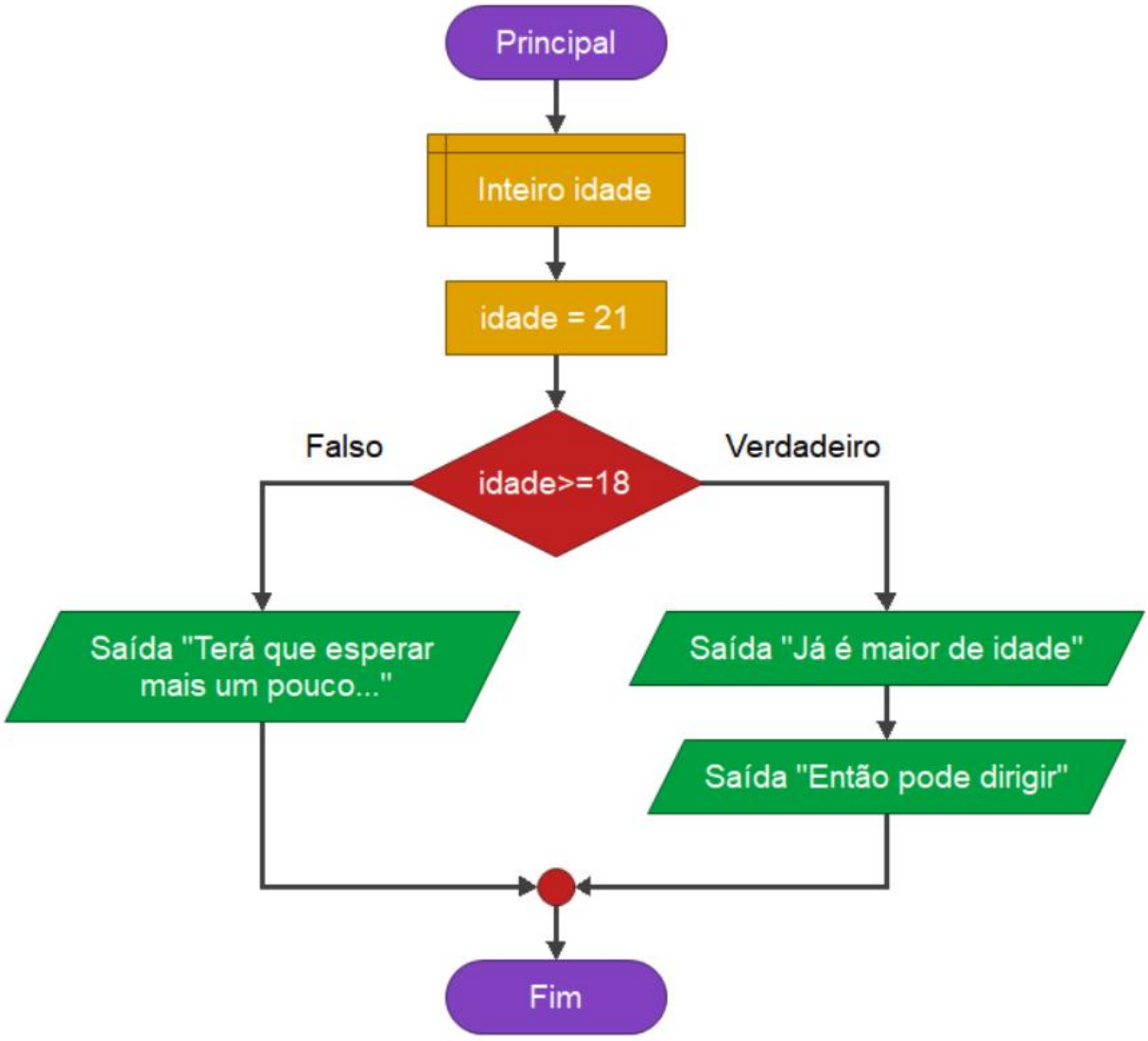
```
int idade = 21;
if (idade>=18){
  printf("Já é maior de idade");
  printf("\nEntão pode dirigir...");
}
```



Perceba que neste caso não há ações a serem feitas caso a expressão seja falsa.

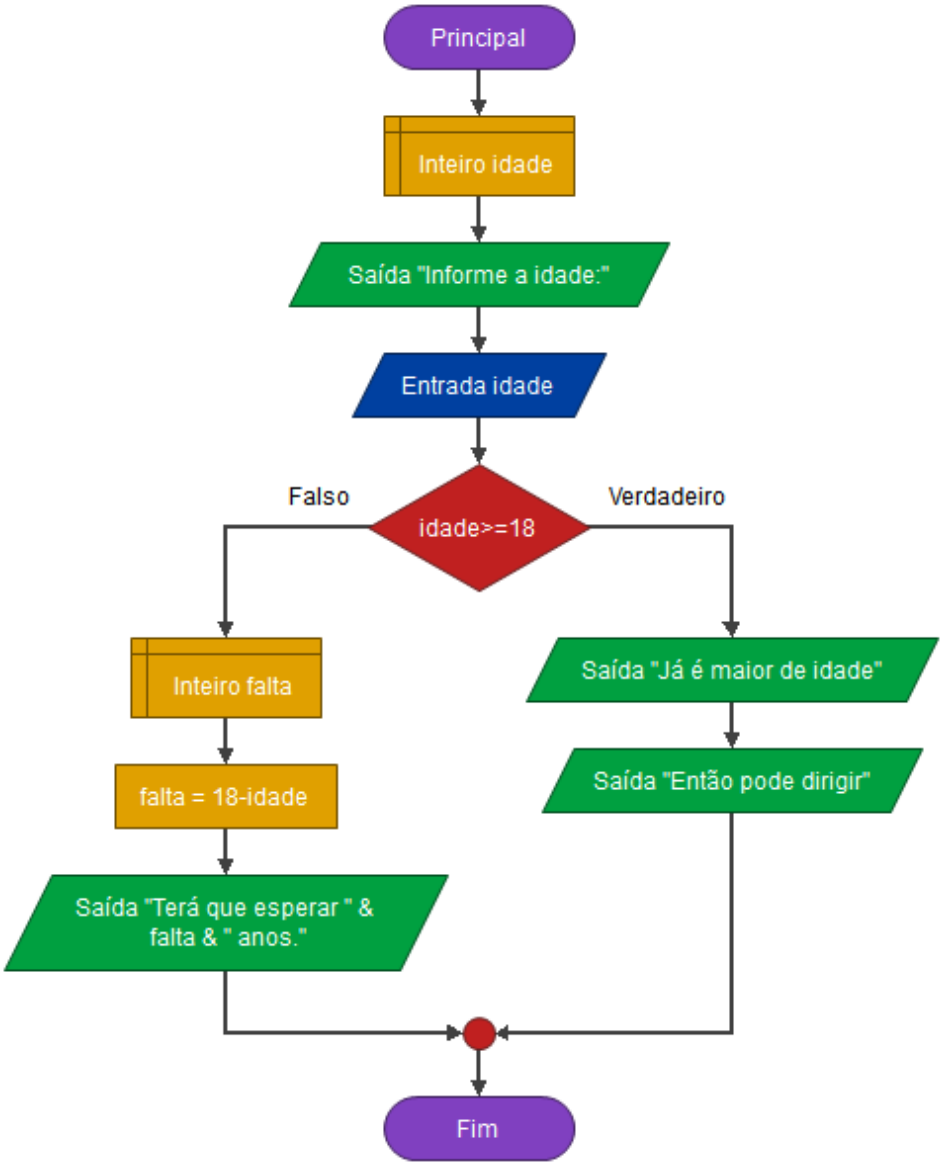
Contudo, é possível, incluir tais ações com a palavra-chave *else*.

```
int idade = 21;
if (idade>=18){
  printf(“Já é maior de idade”);
  printf(“\nEntão pode dirigir...”);
} else {
  printf(“\nTerá que esperar mais um pouco...”);
  //aqui pode ter quantos comandos você quiser...
}
```



Neste caso como a variável *idade* recebe via código o valor 21 então o código dentro do bloco do *else* nunca será executado. Vamos adaptar o código para incluir uma entrada de dados para receber a idade do usuário via *scanf()* e, caso menor de idade, informar quantos anos faltam para tirar a carteira de motorista.

```
int idade;
printf("Informe a idade: ");
scanf("%i", &idade);
if (idade>=18){
  printf("Já é maior de idade");
  printf("\nEntão pode dirigir...");
} else {
  int falta = 18-idade;
  printf("\nTerá que esperar %i anos.", falta);
}
```



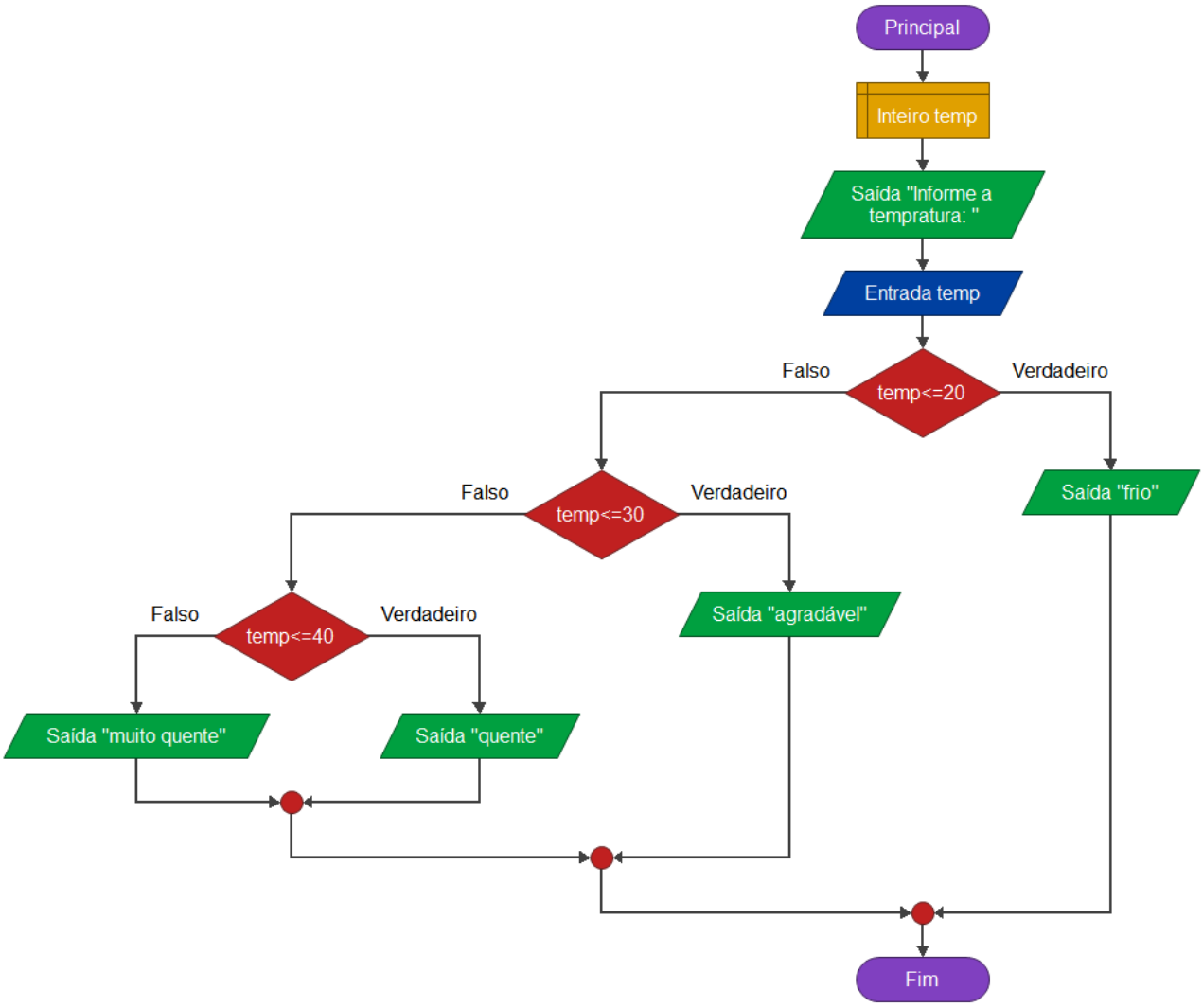
5.2.2. *Decisão* composta

Também é possível alinhar vários *if´s* uns dentro dos outros. Isso permite resolver situações esta: “Leia a temperatura e informe na saída padrão “frio” para temperaturas menores ou iguais a 20 ºC, “agradável” para temperaturas acima de 20 ºC e até 30 ºC, quente para temperaturas acima de 30 ºC até 40 ºC e “muito quente” para temperaturas acima de 40 ºC. O código ficará assim:

```
int temp;
printf(“Informe a temperatura: ”);
scanf(“%i”, &temp);
if (temp<=20){
  printf(“frio”);
} else if (temp<=30){
  printf(“agradável”);
```

```
} else if (temp<=40){
  printf(“quente”);
} else {
  printf(“muito quente”);
}
```

Perceba que se a temperatura for menor que 20 ºC então o fluxo de execução entra no primeiro bloco de código do *if* e não vai mais nem testar as próximas condições. veja:

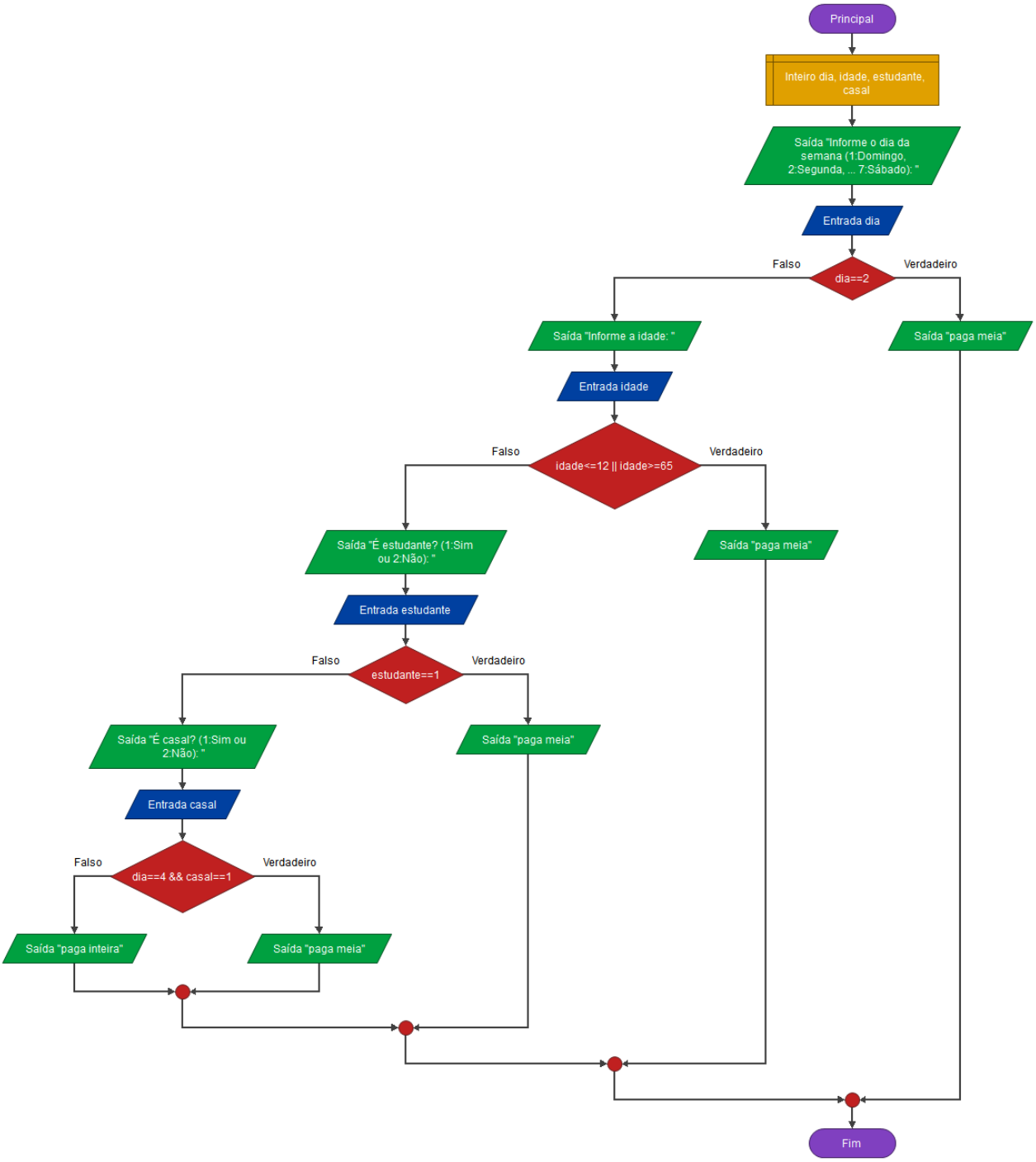


Perceba que não é necessário incluir uma expressão lógica como (temp>20 && temp<=30) na segunda condição, pois temp<=30 já irá atender o que é preciso, pois se o fluxo de execução esta na segunda expressão é porque com certeza o temp é maior que 20.

Agora, para finalizar, vamos fazer um algoritmo para resolver o seguinte problema: Um cinema possui dois tipos de entradas, a meia entrada e entrada inteira. Nas segundas-feiras todos os expectadores pagam meia entrada. Nos outros dias da semana apenas

crianças (12 anos ou menos), estudantes ou idosos com 65 anos ou mais pagam meia entrada. Na quarta-feira casais também pagam meia, ou seja, duas pessoas que declaradamente são um casal e que estão juntas comprando o ingresso também pagam meia entrada cada. Qual seria o algoritmo para resolver isso? Vamos começar pelo fluxograma.

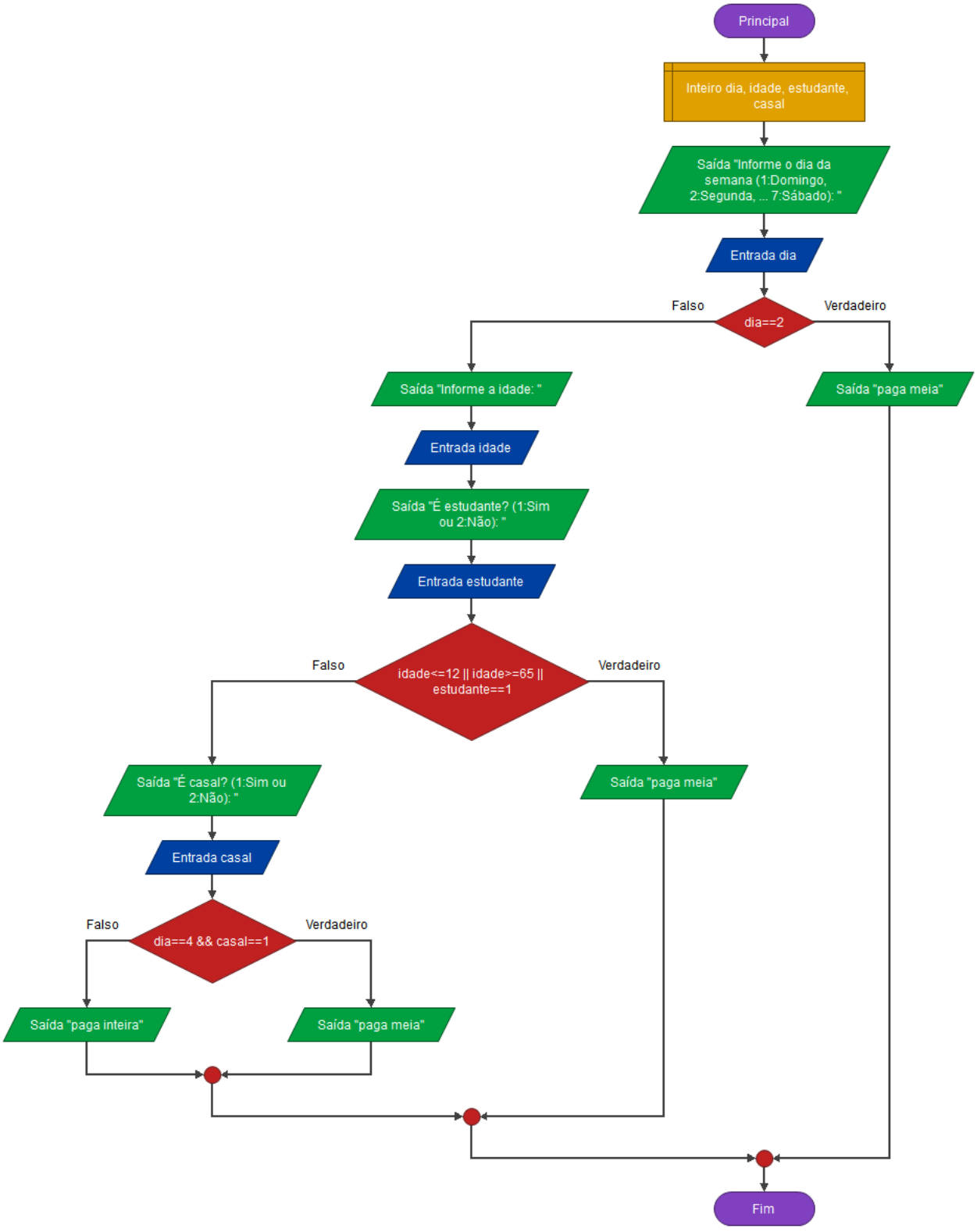
Proposta 1:



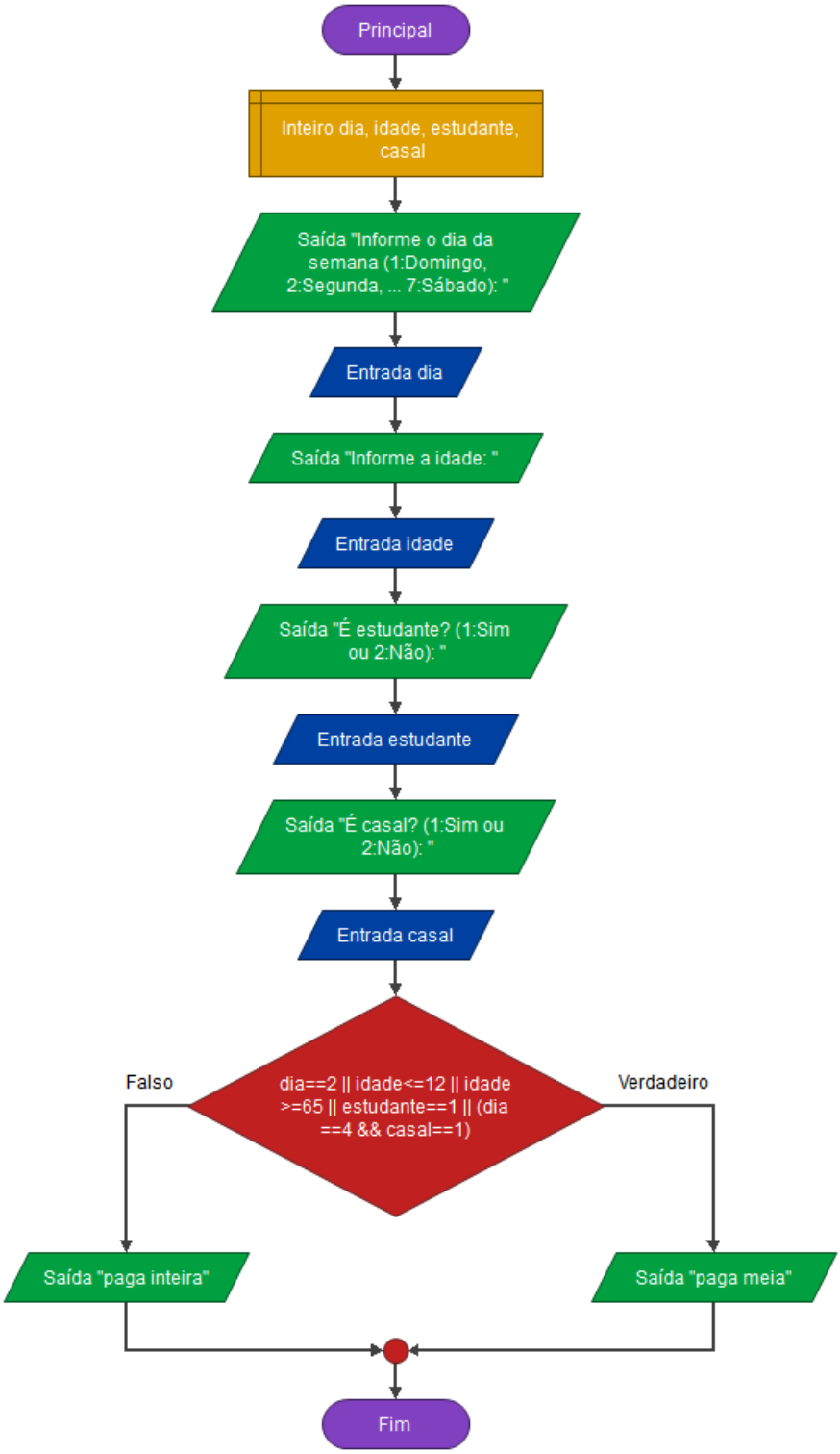
Veja que neste caso as expressões estão todas separadas. Podemos juntar no mesmo *if*

ligando as expressões por ou (||) no caso de idosos, crianças e estudantes, veja na próxima proposta abaixo.

Proposta 2:



No caso acima, é preciso informar ao sistema a idade e também se a pessoa é estudante antes da decisão ser tomada pelo sistema, então aumentamos uma entrada de dados desnecessária no caso da pessoas ser um idoso por exemplo. Podemos ainda reduzir ainda mais o fluxograma em troca de aumentar a complexidade da expressão principal, veja abaixo a “Proposta 3”:



Aqui a desvantagem é que todas as entradas precisão ser feitas antes da decisão. Isso significa que mesmo que a pessoa seja um estudante, terá ainda que informar a idade e se é casal ou não por exemplo. Isso pode atrasar a bilheteria do cinema 😉. Perceba como utilizar a proposta 1, 2 ou 3 é uma decisão de projeto do software.

O código fonte da proposta 1 será:

```c
#include <stdio.h>
int main() {
    int dia, idade, estudante, casal;
    printf("Informe o dia da semana (1:Domingo, 2:Segunda, ... 7:Sábado): ");
    scanf("%i", &dia);
    if (dia == 2) {
        printf("paga meia");
    } else {
        printf("Informe a idade: ");
        scanf("%i", &idade);
        if (idade <= 12 || idade >= 65) {
            printf("paga meia");
        } else {
            printf("É estudante? (1:Sim ou 2:Não): ");
            scanf("%i", &estudante);
            if (estudante == 1) {
                printf("paga meia");
            } else {
                printf("É casal? (1:Sim ou 2:Não): ");
                scanf("%i", &casal);
                if (dia == 4 && casal == 1) {
                    printf("paga meia");
                } else {
                    printf("paga inteira");
                }
            }
        }
    }
    return 0;
}
```

Já da proposta 3 o código fonte é:

```c
#include <stdio.h>
int main() {
    int dia, idade, estudante, casal;
    printf("Informe o dia da semana (1:Domingo... 7:Sábado): ");
    scanf("%i", &dia);
    printf("Informe a idade: ");
    scanf("%i", &idade);
    printf("É estudante? (1:Sim ou 2:Não): ");
```

```
    scanf("%i", &estudante);
    printf("É casal? (1:Sim ou 2:Não): ");
    scanf("%i", &casal);
    if(dia==2 || idade<=12 || idade>=65 || estudante==1 || (dia==4 && casal==1)){
        printf("paga meia");
    } else {
        printf("paga inteira");
    }
    return 0;
}
```

5.3. COMANDO SWITCH CASE.

Além do comando *if* temos outro comando para seleção múltipla. Normalmente o comando *switch* é menos utilizado que o comando *if* e tudo que é possível fazer com *switch* é possível fazer com *if*, veja um exemplo:

```
#include <stdio.h>
int main(){
  int valor;
  printf ("Digite um valor de 1 a 7: ");
  scanf("%i", &valor);
  switch(valor){
    case 1:
      printf ("Domingo\n");
      break;
    case 2:
      printf ("Segunda\n");
      break;
    // inclua aqui os outros dias da semana...
    default :
      printf ("Valor invalido!\n");
  }
  return 0;
}
```

Outro exemplo é o da calculadora abaixo. Neste caso o caractere é utilizado para a tomada de decisão no switch/case. Perceba que neste caso as aspas simples são utilizadas ao lado de cada case como no exemplo case '+':

```
#include<stdio.h>
#include<stdlib.h>
int main(void)
{
    float num1, num2;
    char op;
```

```
    printf("### Calculadora ### \n");
    printf("Exemplo de expressão: 2+2\n");
    printf("Informe a expressão:");
    scanf("%f%c%f", &num1, &op, &num2);
    switch(op){
    case '+':
     printf("= %.2f\n", num1+num2);
      break;
    case '-':
      printf("= %.2f\n", num1-num2);
      break;
    case '*':
    case 'x':
      printf("= %.2f\n", num1*num2);
      break;
    case '/':
    case':':
      printf("= %.2f\n", num1/num2);
      break;
    default:
      printf("\nErro!");
    }
    return 0;
}
```

5.4. OPERADOR TERNÁRIO

Este comando é uma alternativa para substituir o comando “if...else” em algumas situações simples.

A sintaxe é a seguinte:

```
Condição ? verdadeiro : falso
```

Normalmente a condição vem circundada em parênteses e os comandos a seguir devem ser simples para evitar problemas na leitura posteriormente. Não seguir essas indicações pode gerar um código bem confuso. Veja um exemplo.

```
int x=7;
(x>0) ? puts("x é positivo!") : puts("x é zero ou menor.");
```

Outro exemplo segue abaixo:

```
#include <stdio.h>

int main (void){
```

```
  int n;
  printf("Informe um numero: ");
  scanf("%d",&n);

  n>=0 ? n++ : n--;
  printf("Número: %d", n);

  return 0;
}
```

É possível, mas não recomendado, fazer grandes blocos de código com o comando ternário, ou ainda aninhar vários comandos um dentro do outro como abaixo.

```
#include <stdio.h>
int main (void){
  int n;
  printf("Informe um numero: ");
  scanf("%d",&n);

  (n==0) ? puts("É zero!") : (n>0) ? puts("Maior que zero!") : puts("Menor
que zero!");

  return 0;
}
```

Apesar de executar perfeitamente, o código acima se torna confuso. Evite isso!

5.5. EXERCÍCIOS PROPOSTOS

5.1) Qual é a saída do programa abaixo?

```
int main()
{
    float result, a=1, b=2, c=3, d=7, e=8;
    result=(b-(c*7))/(4-(a*c*a));
    if(result >= -19)
        printf("Ok");
    else
        printf("%.2f", result);
    return 0;
}
```

a) -19.000000
b) -48.000000
c) -22.000000
d) 22.00
e) Ok

5.2) Escreva um programa para ler 2 valores reais (considere que não serão informados valores iguais) e escreva na saída padrão do sistema qual é o maior dos números informados.

5.3) Escreva um programa para ler o ano de nascimento de uma pessoa e escrever uma mensagem que diga se ela poderá ou não votar este ano (não é necessário considerar o mês em que ela nasceu).

5.4) Escreva um programa que verifique a validade de uma senha fornecida pelo suário. A senha válida é o número 1234. Devem ser impressas as seguintes mensagens:

```
ACESSO PERMITIDO caso a senha seja válida.
ACESSO NEGADO caso a senha seja inválida.
```

5.5) Faça um programa que lê um número inteiro de 1 a 7 e imprime o dia da semana correspondente (1=Domingo, 2=segunda...), se o número estiver fora deste intervalo o programa deve imprimir uma mensagem de erro.

5.6) Faça um programa que lê três números do teclado, descobre qual deles é o menor e imprime este número na tela.

5.7) Faça um programa que lê três notas (números reais), calcula a média entre elas, imprime a média, se a média é menor que 7 imprime “Aluno em exame!” e se não imprime “Aluno aprovado!”.

5.8) Faça um programa que lê três notas (números reais), calcula a média entre elas, imprime a média, se a média é menor que 7 imprime “Aluno em exame!” e se não imprime “Aluno aprovado!”. Se o aluno estiver em exame, peça para o usuário informar a nota do exame, calcule a fórmula (média das notas + nota exame)/2. Se o resultado for maior ou igual a 5 informe “Aluno passou com exame”, senão informe “Aluno reprovado”.

5.9) Escreva abaixo o código de um programa que calcule a média de quilômetros feitos com cada litro de combustível. Solicite a entrada de dados com quilômetros e litros e depois exiba o cálculo da média e caso seja menor que 8 km/l informe que é preciso trocar o carro por um mais econômico.

5.10) Tendo como entrada a altura e o sexo (codificado da seguinte forma: 1: feminino 2: masculino) de uma pessoa, construa um programa que calcule e imprima seu peso ideal, utilizando as seguintes fórmulas:

- para homens: (72.7 * Altura) – 58

- para mulheres: (62.1 * Altura) – 44.7

5.11) Escreva um programa que leia as medidas dos lados de um triângulo e escreva se ele é Equilátero, Isósceles ou Escaleno. Sendo que:

- Triângulo Equilátero: possui os 3 lados iguais.
- Triângulo Isóscele: possui 2 lados iguais.
- Triângulo Escaleno: possui 3 lados diferentes.

5.12) Escreva um programa que leia o valor de 3 ângulos de um triângulo e escreva se o triângulo é Acutângulo, Retângulo ou Obtusângulo. Sendo que:

- Triângulo Retângulo: possui um ângulo reto. (igual a 90º)
- Triângulo Obtusângulo: possui um ângulo obtuso. (maior que 90º)
- Triângulo Acutângulo: possui três ângulos agudos. (menor que 90º)

5.13) Escreva um programa para ler o número de lados de um polígono regular e a medida do lado (em cm). Calcular e imprimir o seguinte:

- Se o número de lados for igual a 3 escrever TRIÂNGULO e o valor da área
- Se o número de lados for igual a 4 escrever QUADRADO e o valor da sua área.
- Se o número de lados for igual a 5 escrever PENTÁGONO.
- Caso o número de lados seja superior a 5 escrever POLÍGONO NÃO IDENTIFICADO.

5.14) Quais expressões abaixo resultariam em resultado verdadeiro “true” se incluídas em um comando condicional “if” após a seguinte declaração de variáveis: int x=0, y=10;

a) x<10 && y>10
b) x<=10 && y<10
c) x<0 || y>10
d) x>0 && y>=10
e) x<=0 || y<10

5.15) Escreva um programa para ler 3 valores inteiros (considere que não serão lidos valores iguais) e escrevê-los em ordem crescente.

5.16) Faça um programa que pede ao operador para digitar a temperatura ambiente e utilizando comandos “if” ou “if-else”, faz o seguinte. Se a temperatura for maior ou igual a 40, imprime “Muito quente”, se a temperatura menor que 40 e maior ou igual a 30, imprime “Quente”, se a temperatura for menor que 30 e maior ou igual a 20, imprime “Agradável” e se a temperatura for menor que 20, imprime “Frio”.

5.17) Escreva um programa para ler 2 valores inteiros e uma das seguintes operações a serem executadas (codificada da seguinte forma: 1. Adição, 2. Subtração, 3. Divisão, 4. Multiplicação). Calcular e escrever o resultado dessa operação sobre os dois valores lidos. Observação: Considere que só serão lidas as operações 1, 2, 3 ou 4.

5.18) Faça um programa utilizando o comando “switch”. O programa deve se comportar como uma calculadora, onde o operador escolhe entre as operações soma (+), subtração (-), multiplicação (*), divisão (/), potência (^), e raiz quadrada (r). O programa

deve apresentar todas as opções no início e o operador escolhe a operação desejada digitando o símbolo. O programa deve então ler os dois números e realizar a operação.

5.19) Verifique o programa abaixo:

```
#include <stdio.h>
int main(){
    int a, b, c;
    printf("digite a:");
    scanf("%d", &a);
    printf("digite b:");
    scanf("%d", &b);
    printf("digite c:");
    scanf("%d", &c);
    if (a >= b){
        if (a >= c && b >= c)
            printf("%d %d %d\n", c, b, a):
        if (a >= c && b <= c)
            printf('%d %d %d\n", b, c, a);
    }
    if (b >= a){
        if (b >= c && a >= c)
            printf('%d %d %d\n", c, a, b);
        if (b >= c && a <= c)
            printf("%d %d %d\n", a, c, b):
    }
    if (c >= a){
        if (c >= b && b >= a)
            printf("%d %d %d\n', a, b, c);
        if (c >= b && b <= a)
            printf('%d %d %d\n", b, a, c);
    }
    return 0;
}
```

O que será impresso por esse programa:

a) O programa funciona e imprime os números na ordem decrescente;
b) O programa funciona e imprime o menor número;
c) O programa funciona e imprime os números na ordem decrescente;
d) O programa funciona e imprime o maior número;
e) Nenhuma das alternativas;

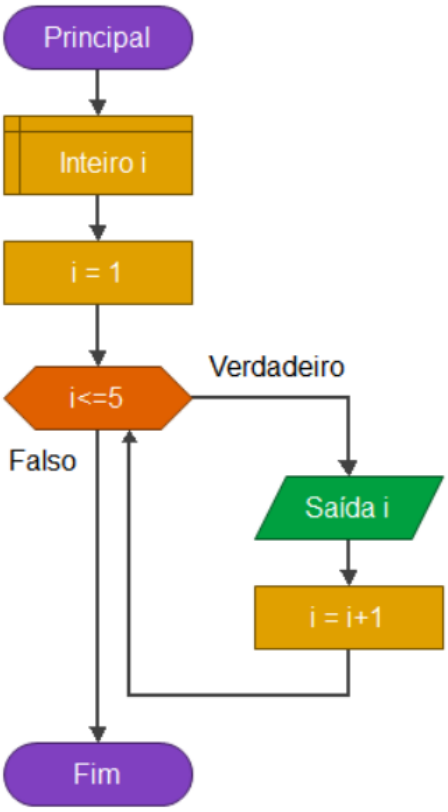
5.20) Faça um programa que lê do teclado um número do tipo unsigned char (valores de 0 a 255) e apresenta na tela o valor dos oito bits que compões este número. Por exemplo, de a entrada for 12 a saída deve ser 00001100. Utilize o comando “if – else” para determinar se o usuário realmente informou um valor de 0 a 255 e caso contrário exiba uma mensagem de erro e encerre o programa.

6. Laços de repetição

Existem 3 comandos que executam laços de repetição. Os 3 funcionam da mesma forma, mas tem pequenas variações. Na prática, tudo que você pode fazer com o while você pode fazer com o for ou com o do...while então depois que você conhecer a fundo os 3 comandos verá que a diferença é pequena entre eles.

6.1. COMANDO DE CONTROLE WHILE

É o principal comando para repetição, veja o fluxograma abaixo que mostra como exibir os números de 1 a 5 na saída padrão do sistema via printf().



Em linguagem C o código equivalente é:

```c
int main() {
  int i;
  i=1;
  while(i<=5){
    printf("%i", i);
    i++;
  }
  return 0;
}
```

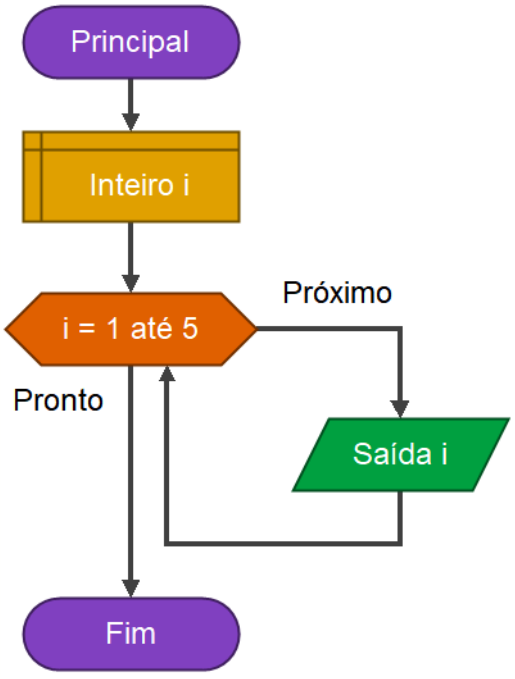
É de fundamental importância incrementar a variável de controle (neste caso a variável i) porque sem isso o laço de repetição entrará em loop, ou seja, ficará executando eternamente.

6.2. COMANDO DE CONTROLE FOR

O mesmo código para impressão dos números de 1 a 5 pode ser feito com o comando *for*, veja:

```
#include <stdio.h>
int main() {
  for(int i=1; i<=5; i++){
    printf("%i", i);
  }

  return 0;
}
```



O comando for possui 3 argumentos for(arg1; arg2; arg3) separados por ';' e que são opcionais com exceção do arg2.

O arg1 é utilizado para criação e definição da variável de controle e só é executado uma vez no início do processamento do comando *for*.

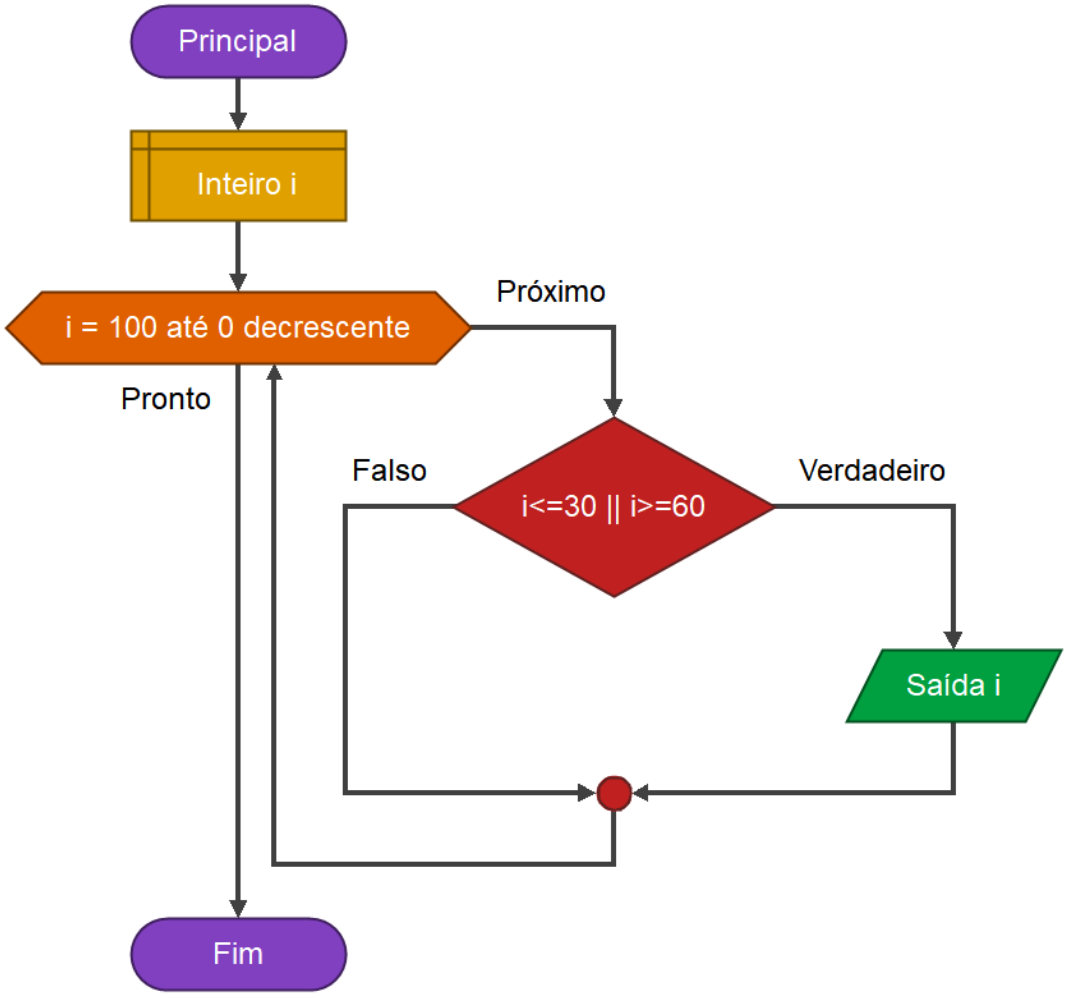
O arg2 é a condição, ou seja, o bloco de código do comando *for* só executa se a condição for verdadeira. A condição, ou seja, o arg2 é verificado antes de cada interação.

O arg3 é usado para incrementar ou decrementar a variável de controle e é executado sempre após a execução do bloco de código.

Os argumentos 1 e 3 são opcionais.

Abaixo temos um exemplo de código que imprime os números de 100 até 0 em ordem decrescente, com exceção dos números entre 30 e 60 que não são impressos.

```c
#include <stdio.h>
int main() {
  for(int i=100;i>=0;i--){
    if(i<=30 || i>=60){
      printf("\n%i", i);
    }
  }
  return 0;
}
```

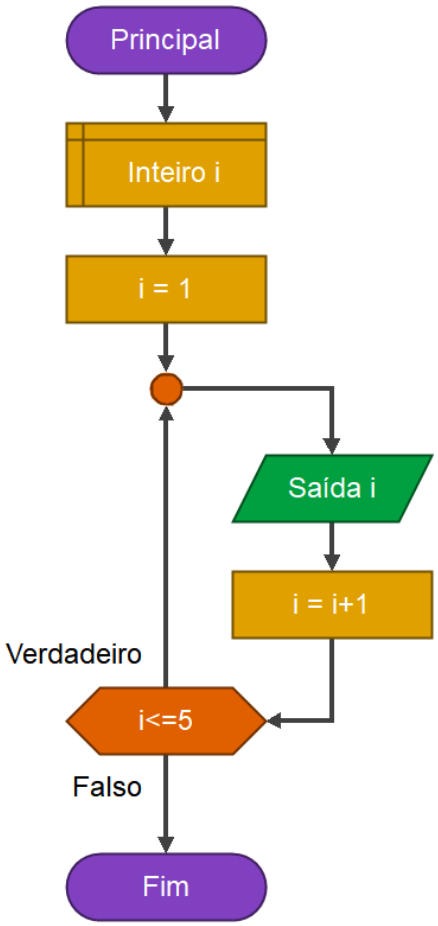


Ao executar o código acima perceba que o número 60 é precedido do 30 e nada entre eles é impresso.

6.3. O COMANDO “DO – WHILE”

Já o comando *do...while* é muito parecido com o while com excessão da primeira interação. Perceba que tanto no while como no for se a condição testada for falsa o comando simplesmente encerra sem executar o bloco de código nenhuma vez. Já no do...while o bloco de código é executado ao menos uma vez, pois a condição só é testada depois da primeira execução, veja:

```c
#include <stdio.h>
int main() {
  int i;
  i=1;
  do {
    printf("%i", i);
    i++;
  } while(i<=5);
  return 0;
}
```

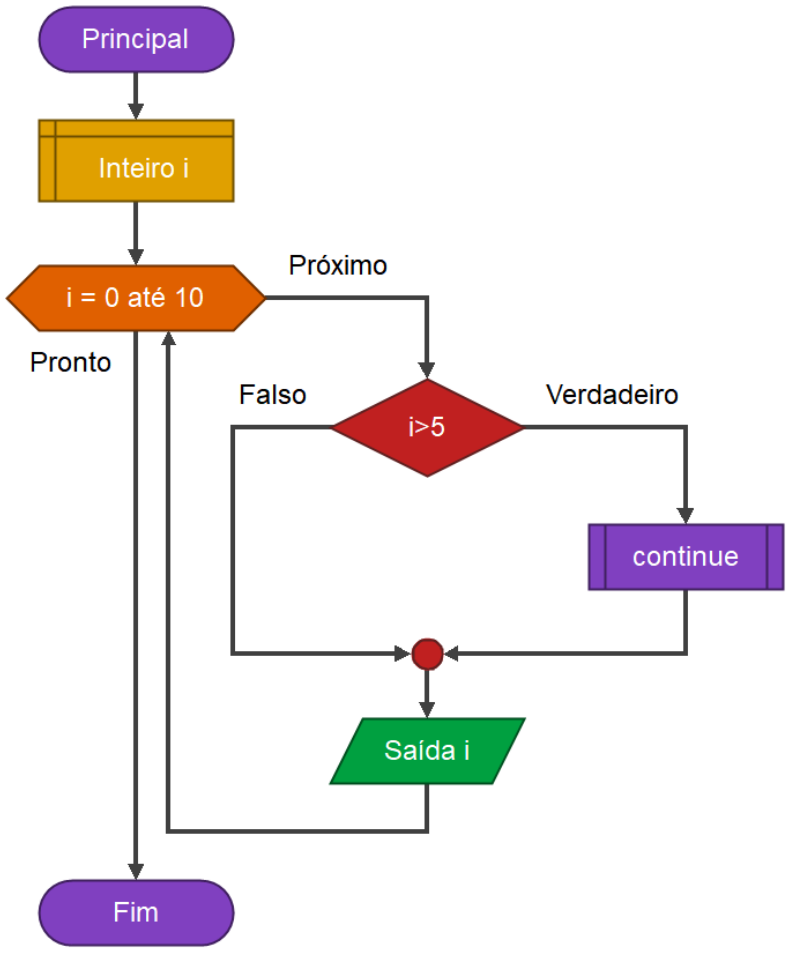


No fluxograma perceba que a condição *(i<=5)* só é verificada depois que a primeira impressão e incremento são executados.

6.4. BREAK E CONTINUE

Existem 2 palavras chave especiais que podem ser usadas com qualquer laço de repetição e que alteração seu fluxo de execução. Estamos falando do break e continue. A palavra-chave continue, quando executada tem a função de reiniciar o laço de repetição ignorando os comando seguintes da execução atual. Veja:

```
#include <stdio.h>
int main() {
  for(int i=0;i<=10;i++){
    if(i>5){
       continue;
    }
    printf("\n%i", i);
  }
  return 0;
}
```



O código acima faz com que seja impresso na tela apenas os números de 0 até 5 pois quando o i é maior que 5 o comando continue é acionado reiniciando o fluxo de execução antes que o prinft() seja executado.

Já o comando break encerra completamente a execução do laço de repetição, seja *while, for* ou *do...while*. Veja um exemplo onde a impressão ocorre de 0 até 4 apenas:

```
for(int i=0;i<=10;i++){
    if(i==5){
       break;
    }
    printf("\n%i", i);
}
```

Conforme falamos no capítulo sobre os comandos condicionais, a linguagem C avalia qualquer número diferente de zero como verdadeiro. Portanto, o código acima é válido. Todos os exemplos abaixo são laços de repetição que vão executar até que um comando *break* seja acionado.

```
while(1){
  //seu código aqui!
}
```

O comando *while* acima irá executar até algum *break* ser acionado.

```
for(;1;){
  //seu código aqui!
}
```

O primeiro e o terceiro argumento do comando for são opcionais. Desta forma, o argumento do meio que é avaliado como expressão lógica será sempre verdadeiro e o laço sempre irá executar até que algum *break* seja acionado.

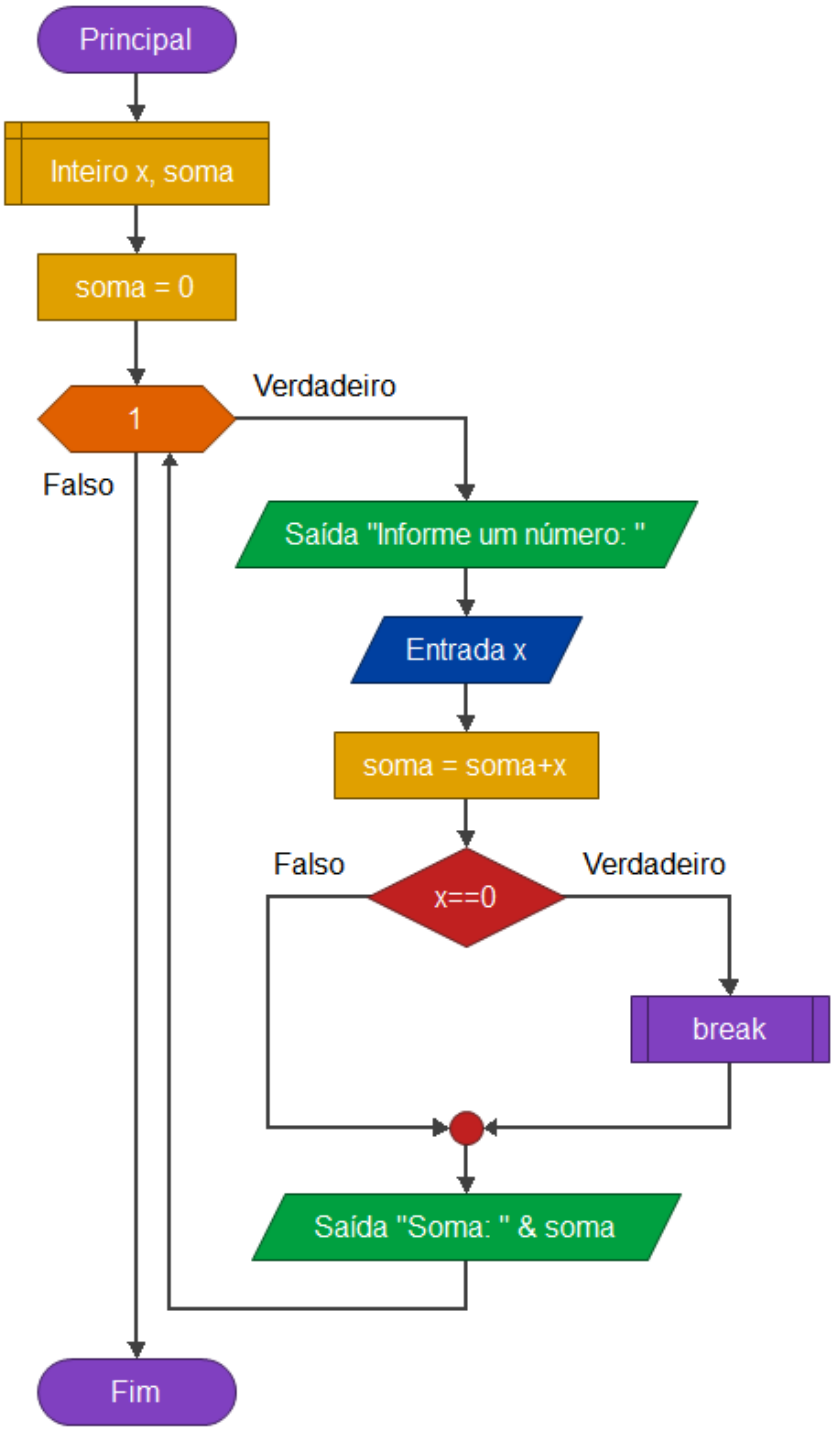
Perceba que *while(1)* é igual a *while(1==1)* por exemplo. Pois tanto 1 quanto 1==1 são consideradas expressões com resultado lógico verdadeiro na linguagem C.

O código abaixo pede para o usuário informar um número. Se o número digitado for diferente de zero o programa acumula o valor na variável soma, exibe a soma na tela e pede para informar outro número. Enquanto o número zero não for informado o programa irá rodar eternamente. Veja o fluxograma deste código em seguida.

```
int main() {
  int x, soma=0;
  while(1){
    printf("Digite um número:");
    scanf("%i", &x);
    soma=soma+x;
```

```
    if(x==0)
       break;
    printf("Soma: %i\n", soma);
  }
  return 0;
}
```

O fluxograma equivalente segue abaixo:



6.5. EXERCÍCIOS PROPOSTOS

6.1) Faça um programa utilizando o comando “for” que apresenta a sequência de números de 5 (inclusive) até 20 (inclusive). Refaça o código utilizando o comando “while”.

6.2) Modifique o programa anterior para que a sequência apresentada seja de 20 (inclusive) a 5 (inclusive).

6.3) Faça um programa que lê dois números inteiros do teclado e testa se o primeiro é menor que o segundo. Em caso afirmativo imprima todos os números do intervalo entre eles.

6.4) Escreva um programa que imprima todos os números pares entre 0 e 50 e em seguida imprima todos os números ímpares. Deixe um espaço entre os números.

6.5) Para que a divisão entre 2 números possa ser realizada, o divisor não pode ser nulo (zero). Escreva um programa para ler 2 valores e imprimir o resultado da divisão do primeiro pelo segundo. OBS: O programa deve validar a leitura do segundo valor (que não deve ser nulo). Enquanto for fornecido um valor nulo a leitura deve ser repetida. Utilize a estrutura de repetição na construção da repetição de validação.

6.6) Escreva abaixo um programa que imprime de 1 (inclusive) a 100 (inclusive) na tela, mas que não imprime o número 50, ou seja, após imprimir o 49 a próxima saída deverá ser 51.

6.7) Utilizando o comando “for”, faça um programa que imprime de 1 a 10 mil na tela, um abaixo do outro, com exceção dos que são divisíveis por 10.

6.8) Crie um programa que recebe números inteiros do usuário, imprime o quadrado do número recebido e solicita o próximo número. Caso o número informado seja zero o programa deve finalizar.

6.9) Receba um número inteiro positivo n, calcule e exiba a soma dos n primeiros números naturais. O conjunto dos números naturais é N = {0, 1, 2, 3, 4, 5, 6, 7...}, portanto, se o usuário informar 4 o resultado deverá ser 6, ou seja, a soma de 0, 1, 2 e 3.

6.10) Receba um número inteiro positivo n e imprima os n primeiros naturais ímpares. Exemplo: Para n=4 a saída deverá ser 1,3,5,7.

6.11) Dados n e uma sequência de n números inteiros positivos, determinar a soma dos números pares.

6.12) Escreva abaixo o código de um programa que calcule a média de quilômetros feitos com cada litro de combustível de um determinado percurso. O programa deve solicitar a entrada de quilômetros e litros e depois exibir o cálculo informando que o usuário deve procurar um carro “mais econômico” caso a média seja inferior a 10

km/litro. Após o primeiro calculo o programa deve solicitar ao usuário que informe o número 1 para executar um novo cálculo ou 0 para sair do programa.

6.13) Faça um programa utilizando ‘while’ que apresenta a tabuada do 7 na tela.

6.14) Faça um programa utilizando ‘for’ que apresenta a tabuada do 3 na tela.

6.15) Faça um programa que apresenta as tabuadas de 1 a 10 utilizando dois laços de repetição um dentro do outro. Utilize ‘while’.

6.16) Faça um programa que apresenta as tabuadas de 1 a 10 utilizando dois laços de repetição um dentro do outro. Utilize ‘for’.

6.17) Utilizando os comandos “for” e “if”, faça um programa que lê 5 valores do teclado e informa quantos deles são negativos e quantos são positivos.

6.18) Escreva um programa que lê 10 números reais. O programa deve imprimir a média, o maior e o menor dos números.

6.19) Faça um programa utilizando o comando “while”, que fica lendo números inteiros do teclado e só finaliza quando o número digitado for 10 ou após 5 iterações, ou seja, após 5 números serem digitados.

6.20) Utilizando o comando “while” faça um programa que calcula o valor de x elevado a y, onde x é um número real lido do teclado e y é um número inteiro lido do teclado.

6.21) Faça um programa que apresenta na tela toda a tabela ASCII, ou seja, apresenta a tela todos os caracteres cujos valores estão entre 0 e 255.

6.22) Escreva um programa que lê um número inteiro do teclado e imprime todos os seus divisores. Divisores são números que permite a divisão com resto igual a zero.

6.23) Faça um programa que lê um número real do teclado e utilizando o comando “while” calcula o fatorial deste número. (ex: 5! é igual a 5 * 4 * 3 * 2 * 1).

6.24) Utilizando o comando “for”, faça um programa que imprime de 1 a 10 mil na tela, um abaixo do outro, com exceção dos que são divisíveis por 33.

6.25) As laranjas custam R$ 0,60 cada se forem compradas menos do que uma dúzia, e R$ 0,50 se forem compradas pelo menos doze. Escreva um programa que leia o número de laranjas compradas, calcule e escreva o valor total da compra. Ao final, peça se o usuário quer realizar novo cálculo.

6.26) Faça um programa que imprime a figura abaixo utilizando laços de repetição. É importante utilizar os comandos “printf("*");” ou “printf("\n");” que devem ser repetidos várias vezes de forma lógica para que a figura seja formada exatamente conforme

abaixo. Utilize quantos laços de repetição achar necessário. Uma sugestão é utilizar dois laços um dentro do outro.

```
*
**
***
****
*****
```

6.27) Altere o programa anterior para imprimir a figura abaixo. Como sugestão gere sequências 1 2 3 4 5 (para números) e 4 3 2 1 0 (para *).

```
****1
***22
**333
*4444
55555
```

6.28) Faça um programa que lê um número do teclado, e utilizando dois comandos “while” apresenta na tela a seguinte matriz, onde n é o número lido e cada elemento da matriz é o resultado da operação indicada. A matriz deve possuir o valor zero nas primeiras e últimas linhas e colunas.

```
N+1 N+2 N+3 N+4
N+2 N+3 N+4 N+5
N+3 N+4 N+5 N+6
N+4 N+5 N+6 N+7


Exemplo: para n = 10, temos:
0     0    0    0
0    13   14    0
0    14   15    0
0     0    0    0
```

6.29) Crie um programa para verificar se um número dado é primo. Utilize apenas números inteiros.

6.30) Escrever um programa que calcula e escreve a soma dos números primos entre 92 e 1478.

6.31) Escreva um método que recebe dois números reais a e b e retorna a soma de todos os números primos existentes entre esses dois.

6.32) Escrever um programa que lê um valor i inteiro e positivo e que calcula e escreve o valor da constante de Euler (e), com aproximação de i termos. A formula a seguir calcula o valor de (e), e o resultado deve ser 2,718282. O programa deve ficar se repetindo enquanto o operador não escolher i = 0. A fórmula está abaixo:

$$e = \sum_{i=0}^{\infty} \frac{1}{i!}$$

6.33) Faça um programa utilizando o comando “while”, que recebe uma entrada que é o valor do lucro de um investidor em Fundo de Renda Fixa e calcula como saída o imposto de renda (IR) que é de 20% sobre o lucro. Ao final peça se o usuário quer realizar novamente o cálculo informando ‘s’ para continuar ou ‘n’ para sair.

6.34) Escreva um programa para imprimir as letras de A a Z.

6.35) Escreva um programa para imprimir as letras de AA a ZZ na mesma sequência existente em planilhas eletrônicas como Microsoft Excel ou LibreOffice Calc.

6.36) Faça um programa utilizando o comando “do – while”, este programa deve ficar lendo uma letra por vez do teclado até que o operador digite a letra x. Após o operador digitar x o programa deve apresentar na tela o número de vezes que o operador digitou dois caracteres iguais em sequência.

6.37) Faça um programa que apresente quatro opções: (a) consulta saldo, (b) saque, (c) depósito e (d) sair. O saldo deve iniciar em R$ 0,00. A cada saque ou depósito o valor do saldo deve ser atualizado e apresentado na tela.

6.38) Faça um programa utilizando o comando “switch”. O programa deve se comportar como uma calculadora, onde o operador escolhe entre as operações soma (+), subtração (-), multiplicação (*), divisão (/), potência (^), e raiz quadrada (r). O programa deve apresentar todas as opções no início e o operador escolhe a operação desejada digitando o símbolo entre parênteses. O programa deve então ler os dois números, realizar a operação e reiniciar o processo até que o operador escolha uma operação inválida (qualquer outro símbolo ou letra).

6.39) Dados n e dois números inteiros positivos i e j diferentes de 0, imprimir em ordem crescente os n primeiros naturais que são múltiplos de i ou de j e/ou de ambos. Exemplo: Para n = 6, i = 2 e j = 3 a saída deverá ser: 0,2,3,4,6,8.

6.40) Dizemos que um inteiro positivo n é perfeito se for igual à soma de seus divisores positivos diferentes de n. Exemplo: 6 é perfeito, pois 1+2+3 = 6. Faça um programa que receba um inteiro positivo n e verifique se n é perfeito.

6.41) Escreva um programa que apresente a série Fibonacci até o décimo sétimo termo. A sequência de Fibonacci é uma sucessão de números obtidos pela soma dos anteriores, a fórmula é sintetizada conforme segue:

$$\begin{cases} 0, \text{ se n=0} \\ 1, \text{se n=0} \\ F(n-1)+F(n-2), \text{outros casos} \end{cases}$$

Resultante em 0, 1, 1, 2, 3, 5, 8, 13, 21...

6.42) O matemático alemão Gottfried Leibniz estabeleceu a fórmula matemática para aproximar o valor de π:

$$\pi = 4 \times \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} \ldots$$

Em notação de somatório:

$$\pi = 4 \times \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

Faça um programa que aproxime o valor de PI (como não é possível calcular até o infinito, solicite que o usuário informe o número de termos. Depois faça 4 testes com seu programa informando 5, 10, 20 e 100 termos para verificar o comportamento do resultado.

6.43) Qual sequência de comandos abaixo, quando executada, resulta no processamento de 10 repetições dos comandos dentro do bloco { ... }?

a) int i=0; while(i++<9) { ... }
b) int i=1; while(i++<9) { ... }
c) int i=0; while(i++<10) { ... }
d) int i=0; while(i++<=10) { ... }
e) int i=1; while(++i<=10) { ... }

6.44) Em relação a saída do programa abaixo se informadas as notas: 5, 7 e 9, assinale a alternativa correta:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
```

```
    //Media de 3 notas
    float n1, n2, n3, media=0;
    puts("Digite a primeira nota:");
    scanf("%f",&n1);
    puts("Digite a segunda nota:");
    scanf("%f",&n2);
    puts("Digite a terceira nota:");
    scanf("%f",&n3);
    if (media > 7) {
       media = (n1 + n2 + n3) / 3;
       printf("Aprovado. Media: %f", media);
  } else {
       printf("Reprovado. Media: %f", media);
  }
    return 0;
}
```

Informe a alternativa correta:

a) Imprime “Aprovado. Media: 7.000000”
b) Imprime “Reprovado. Media: 7.000000”
c) Imprime “Aprovado. Media: 0.000000”
d) Imprime “Reprovado. Media: 0.000000”
e) Nenhuma das alternativas anteriores.

6.45) Considere o seguinte programa:

```
#include <stdio.h>
#include <stdlib.h>
int main(){
   int a=7;
   for(a=7;a>8;--a)
      printf("%d", a);
   if(a>7)
      printf("%d", --a);
   return 0;
}
```

O que será impresso por este programa?

a) 78
b) 7
c) 67
d) 6
e) Nada é impresso na tela

6.46) O código abaixo compila e executa sem erros de sintaxe ou compilação:

```
int main(){
    int n = 0;
    puts("Digite um numero");
    scanf("%d", &n);
    printf("\n");
    int linha=0, coluna=0;
    int temp=0;
    while(linha<4)
    {
        while(coluna<4)
        {
            if(linha==coluna)
            {
                printf("%d  ", n);
                coluna++;
            }
        }
        printf("\n");
        coluna=0;
        linha++;
    }
    return 0;
}
```

O resultado da execução deste código será:

a) Será impresso o número digitado 16 vezes.

b) Será impresso o número digitado 4 vezes, somente quando coluna e linha forem iguais.
c) O código entra em looping na linha 10, pois a variável coluna nunca será incrementada, e portanto, nunca terá um valor igual a linha.
d) Nada será impresso porque a variável linha sempre será maior que a variável coluna.
e) Nada será impresso porque a variável coluna sempre será maior que a variável linha.

6.47) O código abaixo compila e executa sem erros de sintaxe ou compilação:

```
int main(){
    puts("Informe o numero: ");
    int num=0, i=0;
    scanf("%d", &num);
    while(i<10000)
    {
        num = num*i;
        i++;
        if(num>5){
            break;
        }
    }
    printf("Resultado: %d", num);
}
```

Qual será o resultado da execução deste código caso o usuário informe o número 5?

a) “Resultado: 0” com a saída do laço de repetição através da linha 10 no comando “break”.
b) “Resultado: 0” com a saída do laço de repetição após 10 mil execuções”.
c) “Resultado: 75” com a saída do laço de repetição através da linha 10 no comando “break”.
d) Programa entra em looping pois nunca alcança a linha 10 no comando “break”.
e) “Resultado: 0” com a saída do laço de repetição através da linha 10 no comando “break”.

6.48) O código abaixo compila e executa sem erros de sintaxe ou compilação, o resultado será:

```
    int n1, n2;
```

```
puts("Digite o primeiro numero: ");
scanf("%d", &n1);
puts("Digite o segundo numero: ");
scanf("%d", &n2);
if(n1<n2){
     int temp=++n1+1;
     while(temp<n2){
          printf("%d  ", temp);
          temp++;
     }
}
```

Informe a alternativa correta:

a) Impressão em tela dos números do intervalo entre n1 e n2 de 1 em 1.
b) Impressão em tela dos números do intervalo entre n1 e n2 de 2 em 2.
c) Impressão em tela dos números do intervalo entre n1 e n2 de 3 em 3.
d) O programa não irá imprimir nada na tela.
e) Nenhuma das alternativas.

6.49) O código abaixo compila e executa sem erros de sintaxe ou compilação, em relação ao resultado da execução do programa assinale a alternativa correta:

```
float temp=0.0;
while(1){
     scanf("%f", &temp);
     printf("O numero informado: %.2f\n\n", temp);
     if(temp==10 && temp==15){
        printf("Saindo...");
        break;
     }
}
```

Informe a alternativa correta:

a) Se informado 12 o programa é encerrado.
b) Se informado 10 o programa é encerrado.
c) Se informado entre 10 (inclusive) e 15 (inclusive) o programa é encerrado.
d) Se informado entre 11 (inclusive) e 14 (inclusive) o programa é encerrado.
e) Nenhuma das alternativas anteriores.

6.50) O código abaixo compila e executa sem erros de sintaxe ou compilação, em relação ao resultado da execução do programa assinale a alternativa correta:

```
float temp=0.0;
scanf("%f", &temp);
while(temp>=0){
     printf("%.2f\n\n", temp);
     if(temp>=10 && temp<=15){
         printf("Saindo...");
     }
     temp+=2;
}
```

Informe a alternativa correta:

a) Se informado 12 o programa é encerrado sem mostrar a mensagem “Saindo...”.
b) Se informado 10 o programa entra em looping eterno.
c) Se informado entre 10 e depois 15 o programa encerra sem mostrar a mensagem “Saindo...”.
d) Se informado entre 11 (inclusive) e 14 (inclusive) o programa é encerrado mostrando a mensagem “Saindo...”.
e) Nenhuma das alternativas anteriores.

6.51) Qual é a saída do programa abaixo?

```
int main() {
     float result=0, a=1, b=2, c=3, d=7, e=8;
     while(a++>=1 && ++b<=2)
         result=(b-(c*7))/(4-(a*c*a));
     printf("%f",result);
     return 0;
}
```

Informe a alternativa correta:

a) -19.000000
b) -48.000000
c) -22.000000
d) 0.000000
e) 22.000000

6.52) Qual é a saída do programa abaixo?

```
int main() {
     int i=1, i2=18;
     while(1){
     printf("%f", ((++i+i2++)/2.0));
     if(i==3){
        break;
     }
  }
     return 0;
}
```

Informe a alternativa correta:

a) 12.000000
b) 09.00000010.000000
c) 10.00000011.000000
d) 11.00000012.000000
e) 0.000000

6.53) Leia o programa a seguir:

```
#include <stdio.h>
int main(){
     int i, num, pos=0, neg=0;
     printf("Informe 5 números positivos ou negativos\n");
     for(i=0; i<5; i++){
          scanf("%d", &num);
          if(num>=0)
               pos++;
          else
               neg++;
     }
     printf("Positivos : %d\nNegativos : %d", pos, neg);
     return 0;
}
```

Este programa irá imprimir:

a) um número positivo e um negativo;
b) todos os números que o usuário digitar, podendo serem positivos ou negativos;
c) a quantidade de números positivos e negativos que o usuário digitou;
d) o contrário que o usuário digitou, ou seja, os números positivos serão impressos negativos e os negativos serão impressos positivos;
e) nenhuma das alternativas.

6.54) Analise o programa a seguir:

```
#include <stdio.h>
int main(){
     int numero, num=1, i;
     printf("\nDigite um numero positivo: ");
     scanf("%d",&numero);
     for(i=numero; i>1; i--){
          num=num*i;
     }
     printf("fat: %u\n", num);
     return 0;
}
```

O programa irá imprimir:

a) a tabuada na forma crescente do número digitado pelo usuário;
b) a tabuada na forma decrescente do número digitado pelo usuário;
c) o resultado da multiplicação do número digitado pelo número antecedente a ele;
d) o resultado do fatorial do número digitado;
e) nenhuma das alternativas.

6.55) Qual a função do programa abaixo?

```
int main(){
    int n, par, impar, num, cont;
    printf("Digite o tamanho da sequencia: ");
    scanf("%d", &n)
    par   = 0;
    impar = 0;
```

```
    cont  = 0;
    while (cont < n){
        printf("Digite o %do. numero: ",cont+1);
        scanf("%d", &num);
        if (num%2 == 0)
            par = par + 1;
        else
            impar = impar + 1;
        cont = cont + 1;
    }
    Printf("Sequência com ");
    printf("%d números pares e %d impares.", par, impar);
    return 0;
}
```

Assinale a alternativa correta:

a) O programa cria um valor a partir de uma sequência de números digitados pelo operador, mostra esse na tela e informa ao final se este será par ou ímpar.
b) O programa cria um valor a partir de uma sequência de números digitados pelo operador, mostra esse na tela e informa ao final quantos números pares e quantos números ímpares formam o valor final.
c) O programa analisa a sequência de números digitados pelo operador, e diz quantos números digitados são pares e quantos são ímpares.
d) O programa analisa a sequência de números digitados pelo operador, e diz quais números digitados são pares e quais são ímpares.
e) Nenhuma das alternativas.

6.56) O que o programa irá imprimir?

```
int main(){
    int i;
    printf ("Digite um valor: ");
    scanf ("%d", &i);
    do{
        if (i<=100){
            printf (" O valor de i = %d\n", i);
```

```
        } else {
            printf ("Valor maior que 100");
        }
        i++;
    } while (i<=100);
    return 0;
}
```

Assinale a alternativa correta:

a) Todos os valores de 0 a 100
b) Todos os valores menores do número digitado
c) Todos os valores maiores que o numero digitado
d) O valor digitado e os valores até 100 (inclusive)
e) Nenhuma das alternativas

7. Funções

Uma função é um conjunto de comandos que realiza uma determinada tarefa. A função é projetada e referenciada pelo através de um nome. Assim como as variáveis, não é possível ter duas funções com o mesmo nome. A utilização de funções permite organizar e modularizar um programa.

Se você está começando agora a programar, talvez não faça sentido de imediato a utilidade das funções porque provavelmente você está apenas fazendo programas pequenos (até 100 linhas por exemplo). Contudo, em programas maiores as funções são fundamentais para manter a organização do código e facilitar a manutenção e atualização do código.

Durante o livro já nos deparamos com algumas funções como *printf* e *scanf* por exemplo. Agora veremos como você pode programar suas próprias funções.

7.1. RETORNO DE FUNÇÕES

As funções em Linguagem C podem ou não retornar valores para quem as chamou. Na grande maioria das vezes as funções retornam um valor, por exemplo, a função *pow(x,y)* que calcula a potência de x elevado a y retorna o resultado desta operação matemática. Já funções como *printf()* simplesmente exibem o valor na saída padrão e não retornam nada.

Caso a função não tenha retorno devemos preceder seu nome da palavra-chave *void*, veja:

```
#include <stdio.h>

void linha(){
  for(int i=0;i<10;i++)
    printf("-");
}

int main() {
  linha();
  return 0;
}
```

Neste caso a linha padrão com 10 traços será escrita na tela. A saída gerada é:

```
----------
```

Perceba que o primeiro bloco de código possui a programação da função *linha()* e o segundo bloco é a função *main()* que chama da função *linha()*.

Concordo que uma função como a função *linha()* acima não parece muito útil,

mas pense na seguinte questão. Você é o programador de um sistema grande com 23 relatórios diferentes e em cada relatório existe um separador para facilitar a divisão das subpartes. O divisor atual é uma linha com 10 traços. De repente, seu chefe lhe chama para trocar o divisor de todos os relatórios para 10 asteriscos ao invés de 10 traços. Caso você tenha feito uma função única e chamado a função todas as vezes dentro dos vários relatórios então sua tarefa é simples, basta alterar apenas um lugar do código dentro da função. Caso você tenha espalhado centenas de funções *printf()* no seu código, então você terá muito, muito, ..., muito trabalho.

A utilidade maior das funções aparece quando elas retornam valor, para isso é preciso substituir o *void* pelo tipo de dados (int, float, double, char, etc) que será retornado.

Por exemplo a função pi() abaixo, quando chamada, irá retornar o valor de *pi* com apenas 2 casas decimas.

```
float pi(){
  return 3.14;
}
```

Perceba que 3.14 é uma constante do tipo *float* e por isso o tipo de retorno da função é *float*. Já a função abaixo retorna um inteiro aleatório entre 0 e 9, veja:

```
#include <stdio.h>
#include <time.h> //biblioteca necessária para rand()
#include <stdlib.h> //biblioteca necessária para rand()

int randomiza(){
  return rand()%10; //retorna o resto da divisão por 10, portanto, 0 a 9.
}

int main() {
  srand(time(NULL));//inicializa o gerador de números aleatórios de rand()
  printf("%i", randomiza());
  printf("%i", randomiza());
  printf("%i", randomiza());
  return 0;
}
```

Executar o código acima irá gerar 3 dígitos aleatórios de 0 a 9 na saída padrão. Dentro do bloco de código do corpo da função, ou seja, entre os parêntesis, você pode colocar quantas linhas de códigos ou comandos quiser. Contudo, se o tipo de dados de retorno é ‘int’ como no exemplo acima, é preciso ter uma linha com a palavra-chave *return* seguida de um valor inteiro. Ao alcançar o returno a função termina e o valor é retornado para a chamada. O operador ‘%’ resto da divisão utilizado na função *randomiza()* retorna um *int*.

A primeira linha dentro de main() é “`srand(time(NULL));`” e é necessária para

iniciar o gerador de números aleatórios da função *rand()* mas deve ser executada apenas uma vez durante o código, por isso não está dentro da função *randomiza().*

7.2. PARÂMETROS DE FUNÇÕES

Os parâmetros são responsáveis por levar a informação da chamada da função para dentro do processamento da função. Eles são opcionais e ficam sempre entre os parênteses que vêm depois do nome da função. Os parâmetros também podem ser chamados de argumentos da função.

Os parâmetros são fundamentais pois é a forma que a função tem de receber valores para subsidiar seus cálculos. Os argumentos devem ter nomes com as mesmas regras de nomes de variáveis e devem vir precedidos do tipo de dados.

Para facilitar vamos fazer uma função simples que multiplica um valor recebido por 2. Neste caso perceba que chamamos a função dentro do próprio *printf()*, portanto, o retorno da função *dobro(5)* será calculado como o valor 10 e *printf()* usará o valor 10 para imprimir na tela.

```
#include <stdio.h>

int dobro(int a){ //declaração da função
  return a*2; //retorno
}

int main() {
  printf("%i", dobro(5));
  return 0;
}
```

O resultado na saída padrão é: 10

Neste caso a função tem apenas um parâmetro inteiro chamado ‘a’. O parâmetros é obrigatório na hora da chamada da função que ocorre dentro do *printf()*. Não colocar um parâmetro inteiro na chamada da função gera erro em tempo de compilação.

Em resumo, veja que o tipo de dados de retorno da função vem primeiro, neste caso “int”. Depois vêm o nome da função que obedece às mesmas regras de nome de variável, ou seja, não pode ter espaços, não pode começar com número e deve ter letras minúsculas. Por fim, entre os parênteses vêm os argumentos da função que é a forma da função receber valores para subsidiar seus cálculos. Os argumentos devem ter nomes com as mesmas regras de nomes de variáveis e devem vir precedidos do tipo de dados, caso haja mais de um argumento devem ser separados por vírgulas.

Tanto o retorno de dados quanto os parâmetros são opcionais. Caso a função não tenha retorno, lembre-se que é preciso incluir a palavra-chave *void* como tipo de dados de retorno. Veja um exemplo de função *void* que recebe dois parâmetros, o primeiro (int) é o número de vezes que o segundo (char) será repetido na tela:

```c
#include <stdio.h>

void repete(int n, char c){
  for(int i=0;i<n;i++)
    printf("%c", c);
}

int main() {
  repete(5, 'x');
  printf("\n");
  repete(10, '*');
  printf("\n");
  repete(15, '#');
  return 0;
}
```

A saída gerada é:

```
xxxxx
**********
###############
```

Perceba que não é preciso ter o comando *return* em uma função *void* e normalmente elas são usadas apenas para enviar dados para a saída padrão já que não retornam nenhum valor. Tentar atribuir um valor de retorno de uma função *void* gera erro em tempo de compilação. Por exemplo, "*int a = repete(5, 'x')*" iria gerar erro já que a função *repete()* não retorna nada.

7.3. ESCOPO DE VARIÁVEIS

Uma característica fundamental de qualquer programador é identificar o escopo das variáveis que o código está manipulando. Basicamente o escopo pode ser local ou global.

O escopo global ocorre quando definimos a variável na hierarquia maior dentro do programa. No caso da linguagem C, podemos ter apenas uma função main() que é executada quando o programa é chamado pelo sistema operacional. Se declararmos qualquer variável fora do bloco de código, ou seja, fora do escopo da função main() então dizemos que aquela variável é global. Veja:

```c
#include <stdio.h>

int a=7;//Variável global

int main() {
  printf("%i", a);//será impresso 7 na saída padrão
  return 0;
}
```

Contudo, caso tenhamos criado uma outra variável local dentro do bloco de código da função main() então esta variável é que será utilizada pelo printf(), veja:

```
#include <stdio.h>

int a=7;//Variável global

int main() {
  int a=3;
  printf("%i", a);//será impresso 3 na saída padrão
  return 0;
}
```

Acima será impresso 3 na saída padrão. Mas isso não quer dizer que a variável local substituiu a variável global. Pois se utilizarmos uma função no código conforme abaixo ainda acessaremos o valor global.

```
#include <stdio.h>

int a=7;//Variável global

void f(){
  printf("%i", a);//quando chamada imprime 7 na saída padrão
}

int main() {
  int a=3;
  printf("%i", a);//será impresso 3 na saída padrão
  f();//será impresso 7 na saída padrão
  return 0;
}
```

O resultado do programa acima será 37 impressos na saída padrão.

7.4. EXERCÍCIOS PROPOSTOS

7.1) Escreva uma função "float media(float n1, float n2)" que dá como resultado a média dos valores.

7.2) Escreva uma função que receba dois números inteiros retorne o menor número.

7.3) Faça uma função que recebe um valor inteiro e verifica se o valor é par. A função deve retornar 1 se o número for par e 0 se for ímpar.

7.4) Crie uma função “float validaNota(float n){...}” para validar as notas de um sistema acadêmico. As notas devem ser números reais positivos de 0 a 10. Caso a nota esteja fora deste intervalo a função deve retornar -1 e caso a nota seja válida a função deve

retornar a própria nota.

7.5) Crie uma função “float capturaNota(){...}” para capturar as notas de um sistema acadêmico. Esta função deverá possuir um scanf() dentro dela. As notas devem ser números reais positivos de 0 a 10. Caso a nota esteja fora deste intervalo a função deve informar ao usuário e pedir que ele informe novamente a nota. Quando o usuário informar uma nota válida a função retorna a própria nota.

7.6) Faça uma função que recebe um valor inteiro e verifica se o valor é positivo, negativo ou zero. A função deve retornar 1 para valores positivos, -1 para negativos e 0 para o valor 0.

7.7) Construa uma função “float elevado(float x, int y)”, que calcula x elevado a y. Não utilize a função “pow()” da biblioca “math.h”. Preveja condições de erro (exemplo: y negativo).

7.8) Crie a função “void imprime_espacos (int n_espacos)” e a função “void imprime_asteriscos (int n_asteriscos)” elas devem imprimir na tela, sem pular linha, vários caracteres (espaços e asteriscos, respectivamente), de acordo com o número passado pelo parâmetro.

7.9) Escreva uma função que recebe como parâmetro um inteiro positivo ano e devolve 1 se ano for bissexto, 0 em caso contrário. (Um ano é bissexto se (ano % 4 == 0 && (ano % 100 != 0 || ano % 400 == 0)).)

7.10) Escreva um programa em C que leia 5 valores inteiros e imprima para cada um o seu correspondente valor absoluto. Para obter o valor absoluto do número utilize a função Absoluto especificada abaixo:

a) Nome: Absoluto

b) Descrição: Retorna o valor absoluto do número fornecido.

c) Entrada: int n

d) Saída: (int) O respectivo valor absoluto de n.

7.11) Escreva uma função que calcule e retorne a distância entre dois pontos (x1, y1) e (x2, y2). Todos os números e valores de retorno devem ser do tipo float.

7.12) Escreva uma função “potencia(base, expoente)” que, quando chamada, retorna base*expoente. Por exemplo, potencia(3,4) deve retornar 81. Crie uma outra função para utilizar dentro da função potencia para validar que o expoente é um inteiro maior ou igual a 1.

7.13) Escreva um programa que leia 5 números inteiros positivos (utilize uma função que leia esse número e verifique se ele é positivo). Para cada número informado escrever a soma de seus divisores (exceto ele mesmo). Crie a função “SomaDivisores(...)” para obter a soma. A especificação da função esta abaixo:

a) Nome: SomaDivisores

b) Descrição: Calcula a soma dos divisores do número informado (exceto
ele mesmo).

c) Entrada: Um número inteiro e positivo.

d) Saída: A soma dos divisores.

e) Exemplo: Para o valor 8: 1+2+4 = 7

7.14) Escreva uma função que receba 3 notas de um aluno e uma letra. Se a letra for A então a função retorna a média aritmética das notas do aluno, se for P, a sua média ponderada (pesos: 5, 3 e 2) e se for H, a sua média harmônica.

7.15) Faça um programa que use a função valorPagamento para determinar o valor a ser pago por uma prestação de uma conta. O programa deverá solicitar ao usuário o valor da prestação e o número de dias em atraso e passar estes valores para a função valorPagamento, que calculará o valor a ser pago e devolverá este valor ao programa que a chamou. O programa deverá então exibir o valor a ser pago na tela. Após a execução o programa deverá voltar a pedir outro valor de prestação e assim continuar até que seja informado um valor igual a zero para a prestação. Neste momento o programa deverá ser encerrado, exibindo o relatório do dia, que conterá a quantidade e o valor total de prestações pagas no dia. O cálculo do valor a ser pago é feito da seguinte forma. Para pagamentos sem atraso, cobrar o valor da prestação. Quando houver atraso, cobrar 3% de multa, mais 0,1% de juros por dia de atraso.

8. Vetores e Matrizes

8.1. VETORES

São variáveis que agrupam dados de mesmo tipo e são referenciados pelo mesmo nome juntamente com um índice numérico inteiro e positivo. Em linguagem C o índice do primeiro elemento de qualquer vetor começa em zero[11].

Sintaxe:

```
tipo nome_do_vetor[quantidade_de_itens];
```

Exemplo de declaração de um vetor nomeado como ‘a’, do tipo *int e* com o tamanho máximo de 5 números inteiros:

```
int a[5];
```

Podemos entender vetores como sendo apenas uma coluna de uma planilha eletrônica onde cada “linha” possui apenas um elemento. Neste caso referenciamos as linhas por índices inteiros, sendo que a primeira linha tem índice zero, a segunda índice um e assim por diante. Veja como seria o vetor de 5 elementos declarado acima:

índice	valor
0	<lixo>
1	<lixo>
2	<lixo>
3	<lixo>
4	<lixo>

Após a declaração é possível atribuir valor aos elementos de um vetor. Seja via scanf(...) seja via atribuição direta. Veja:

```
a[0]=22;
a[1]=33;
a[2]=44;
a[3]=66;
a[4]=77;
```

Perceba que até que todos os elementos do vetor tenham recebido um valor, o que esta armazena é apenas lixo que estava presente na memória, ou seja, bits zero ou

[11] Esta além do escopo deste livro explicar os detalhes do endereçamento de memória que utiliza o índice do vetor para armazenar e recuperar os valores dos elementos. Mas a bibliografia é vasta a respeito do assunto e tenho certeza que o leitor encontrará isso na internet.

um que estavam já gravadas na memória alocada para o vetor. Depois da execução das linhas acima o vetor fica assim:

índice	valor
0	22
1	33
2	44
3	66
4	77

Depois de declarado, o vetor deve ser usado sempre junto com o índice que fica entre os colchetes '[indice]'. Veja:

```
int main(void) {
  int a[5];
  a[0]=22;
  a[1]=33;
  a[2]=44;
  a[3]=66;
  a[4]=77;
  printf("%i", a[0]);
  return 0;
}
```

O código imprimirá na saída padrão o valor do primeiro elemento do vetor que é 22. Já no caso do comando `printf("%i", a[2]);` a saída padrão exibirá o valor 44.

Perceba que o primeiro item possui índice zero e o último item de um vetor com cinco elementos terá o índice quatro.

Podemos declarar um vetor diretamente assim:

```
int main(void) {
  int a[5] = {22, 222, 2222, 22222, 42};
  printf("%i\n", a[0]);
  printf("%i\n", a[1]);
  printf("%i\n", a[2]);
  printf("%i\n", a[3]);
  printf("%i", a[4]);
  return 0;
}
```

No exemplo acima, o valor impresso na tela será 42.

A grande vantagem dos vetores é poder acessar seus elementos com índices numéricos então para imprimir todos os cinco elementos acima podemos utilizar o seguinte código:

```c
int main(void) {
  int a[5] = {22, 222, 2222, 22222, 42};
  for(int i=0;i<5;i++){
    printf("%i\n", a[i]);
  }
  return 0;
}
```

Esta é grande vantagem da utilização de vetores pois o código para imprimir um vetor com cinco elementos é mesmo código usado para a impressão de um vetor de cinco mil ou cinco milhões de elementos. O que mudará é apenas o limite máximo da variável de controle do laço de repetição que neste caso é a variável ‘i’.

Quando o número de itens inicializados é menor que o número total de itens do vetor, o valor zero é atribuído aos demais itens, veja:

```c
int main(void) {
  int a[5] = {42, 42};
  for(int i=0;i<5;i++){
    printf("Elemento %i: %i\n", i+1, a[i]);
  }
  return 0;
}
```

Imprimirá na saída padrão:

```
Elemento 1: 42
Elemento 2: 42
Elemento 3: 0
Elemento 4: 0
Elemento 5: 0
```

Perceba que foi adicionado 1 na hora de exibir a posição do elemento no *printf().* Isso melhora a exibição na tela para o usuário pois são exibidos os itens de 1 a 5 e não de 0 a 4.

Outra forma de inicializar os vetores é através de laços de repetição. Abaixo criamos e iniciamos com zero dois vetores de 100 elementos cada. Um deles do tipo *float* e outro do tipo *int*.

```c
int main(void) {
  int idades[100];
  float alturas[100];
  for(int i=0;i<100;i++){
    idades[i]=0;
    alturas[i]=0;
  }
  return 0;
}
```

A grande vantagem de utilizar vetores é utilizar laços de repetição para interagir com os elementos. Veja o código abaixo que converte um número decimal em um binário de até 4 bits.

```
int main(void) {
  unsigned int num;
  short resto[4];
  printf("Informe um número:");
  scanf("%i", &num);
  if(num<0 || num>15){
    printf("Impossivel calcular com 4 bits.");
  } else {
    for(int i=0;i<4;i++){
      resto[i]=num%2;
      num=num/2;
    }
    printf("Bin: ");
    for(int i=3;i>=0;i--){
      printf("%i", resto[i]);
    }
  }
  return 0;
}
```

Perceba agora a diferença no código abaixo que realiza a tarefa para um número binário de 32 bits. O trabalho é 8 vezes maior pois saltamos de 4 para 32 bits mas o código é praticamente o mesmo já que utilizamos laços de repetição e índices.

```
int main(void) {
  printf("Conversão Decimal/Binária com 32 bits\n");
  unsigned long int num;
  short resto[32];
  printf("Informe um número:");
  scanf("%li", &num);
  if(num<0 || num>4294967295){
    printf("Impossivel calcular com 32 bits.");
  } else {
    for(int i=0;i<32;i++){
      resto[i]=num%2;
      num=num/2;
    }
    printf("Bin: ");
    for(int i=31;i>=0;i--){
      printf("%i", resto[i]);
    }
  }
  return 0;
}
```

8.2. MATRIZES

Um vetor é uma matriz de apenas uma dimensão. Podemos entender vetores como sendo apenas uma coluna de uma planilha eletrônica onde cada “linha” possui apenas um elemento. Neste caso referenciamos as linhas por índices inteiros, sendo que a primeira linha tem índice zero, a segunda índice 1 e assim por diante.

índice	valor
0	22
1	33
2	44
3	66
4	77
...	...

Dessa forma, no caso do vetor de inteiros acima, o comando `printf("%i", a[3]);` imprime na saída padrão o valor 66. Já matrizes possuem duas dimensões, então podemos representa-las como se fosse uma planilha eletrônica tradicional, com várias linhas e colunas. Veja:

índice/valor	**0**	**1**	**2**
0	22	45	68
1	33	17	40
2	44	67	14
3	66	12	35
4	77	100	42

Veja que a matriz acima possui 5 linhas e 3 colunas. Para criarmos a matriz acima usamos o comando: `int m[5][3];`

A atribuição ocorre da mesma forma, mas é preciso incluir sempre as duas dimensões:

```
m[0][0]=22; //omitimos as demais linhas
m[4][2]=42;
```

Para acessar seus elementos também sempre incluímos as duas dimensões com os índices de linha e coluna respectivamente. O código abaixo exibiria na saída padrão a sequência: 44 14 45 100 42

```
printf("%i ", m[2][0]);
printf("%i ", m[2][2]);
printf("%i ", m[0][1]);
printf("%i ", m[4][1]);
printf("%i", m[4][2]);
```

A linguagem C permite a criação de matrizes de várias dimensões. Matrizes deste tipo, principalmente as de quatro dimensões ou mais são denominadas *tensors* que é um termo para definir estes grandes agrupamentos de dados. Na matemática, *tensors* ou tensores são entidades geométricas que generalizam o conceito de escalares, vetores e matrizes. Os *tensors* são a base fundamental de tecnologias como *deep learning* e *big data*. Um tensor de quatro dimensões pode, por exemplo, guardar um conjunto de milhares de imagens coloridas para treinamento de uma rede neural profunda (*deep neural network*) para reconhecimento e classificação de imagens.

Matrizes são muito úteis para cálculos matemáticos, processamento de imagens ou utilização de grandes massas de dados. Com a advento da *Internet of Things (IOT)* ou Internet das Coisas a comunicação de vários sensores gera grandes volumes de dados e trabalhar com matrizes é fundamental para utilizar essas informações.

Para percorrer uma matriz de duas dimensões utilizamos dois laços de repetição. O laço externo fará a iteração na primeira dimensão, ou seja, pensando no exemplo anterior, a iteração do laço externo é com as linhas da matriz. Já o laço de repetição inteiro fará a iteração com as colunas. Veja o exemplo de código que captura os valores de uma matriz 3x3 e exibe os valores na tela:

```
#include <stdio.h>
int main(void) {
  int m[3][3];
  printf("Informe os valores:\n");
  for(int i=0;i<3;i++){
    for(int j=0;j<3;j++){
      printf("Elemento [%i][%i]:",i,j);
      scanf("%i", &m[i][j]);
    }
  }
  printf("\nVoce informou:\n");
  for(int x=0;x<3;x++){
    for(int y=0;y<3;y++){
      printf("Elemento [%i][%i]: %i\n",x,y,m[x][y]);
    }
  }

  printf("\nResultado:\n");
  for(int x=0;x<3;x++){
    for(int y=0;y<3;y++){
      printf("%6i ",m[x][y]);
    }
    printf("\n");
  }
  return 0;
}
```

Execute e teste este código. Perceba que para qualquer outro processamento com a matriz pode ser efetuado com os dois laços de repetição aninhados conforme foi mostrado acima.

8.3. STRINGS

Também chamadas de cadeias de caracteres a palavra *string* em inglês significa filamento ou corda. Neste sentido, uma *string* é um conjunto de caracteres dispostos em uma determinada ordem.

Um vetor do tipo char é considerado uma *string* em linguagem C, já que não existe um tipo de dados específico para *strings* como existem em outras linguagens.

A declaração de *strings* ocorre da mesma forma que os vetores de outros tipos, veja:

Para declarar um vetor de inteiros usamos, por exemplo: `int a[5];`

Já para um vetor de números reais usamos, por exemplo: `float a[5];`

Então para um vetor de caracteres, ou seja, para uma *string* usamos: `char a[5];`

É importante lembrar que uma *string* poderá armazena apenas uma palavra, pois cada elemento será um caractere, ou seja, uma letra. Abaixo veja como podemos declarar e inicializar *strings* logo na sua declaração:

```
char nome1[10] = {'B','r','a','s','i','l'};
char nome2[10] = {'M', 'e', 'u', ' ', 'B','r','a','s','i','l'};
```

Acima temos a atribuição tradicional, mas sempre considerando que um caractere deve estar inserido em um par de aspas simples. Abaixo, temos um tipo especial de declaração de *string*, percebe que apenas para vetores de caracteres podemos utilizar este sistema de atribuição e apenas na declaração do vetor, veja:

```
char nome1[10] = "Brasil";
char nome2[10] = "Meu Brasil";
```

Perceba também a diferença nas aspas. Pois uma *string* que é um conjunto de um ou mais caracteres deve estar dentro de aspas duplas, já um caractere deve estar inserido em aspas simples.

O código abaixo cria, atribui valores e exibe na saída padrão uma *string*.

```
#include <stdio.h>
int main(void) {
  char nome[10] = "Brasil";
  printf("%s", nome);
  return 0;
}
```

Em memória uma *string* nada mais é que um vetor, então a *string* “Brasil” declarada com dez posições ficaria assim:

índice	valor
0	'B'
1	'r'
2	'a'
3	's'
4	'i'
5	'l'
6	'\0'
7	'?'
8	'?'
9	'?'

Aqui são dois os pontos importantes a serem observados. Primeiro, o caractere de finalização de *string* ‘\0’ (lê-se barra zero) que é utilizado em todas as strings para sinalizar a finalização. Ele é um caractere especial para este fim específico de informar que a *string* acabou. Isso é muito importante porque dá flexibilidade para trabalharmos com strings de qualquer tamanho. Por exemplo, se desejamos armazenas o nome do usuário no sistema, nunca sabemos antecipadamente quantas letras o nome terá. Neste caso, criarmos uma string maior, por exemplo, 255 caracteres, e usamos apenas o que for necessário.

Veja que os símbolos ‘?’ que estão presentes nos índices 7, 8 e 9 da string acima representam lixo na memória. Este lixo não é exibido pela chamada á função printf() feita na linha no comando `printf("%s", nome);` a pois a função printf() irá imprimir apenas os caracteres da *string* até encontrar o ‘\0’. Lembre-se que é um zero e não a letra ó maiúscula!

Veja abaixo o código que imprime na tela uma *string* letra a letra.

```
char str[10]="Brasil";
for(int i=0;i<10;i++){
  if(str[i]!='\0'){
    printf("%c ", str[i]);
  }
}
```

Saber trabalhar com os caracteres separados das strings é importante por permite realizar uma série de operações. Por exemplo, vamos criar um código que imprima uma *string* separando cada caracteres por espaços.

O resultado será: “B r a s i l” ao invés de “Brasil”. Porém perceba abaixo que a única coisa que mudou em relação ao código acima é que há um espaço logo após a máscara de impressão de caracteres “%c “ na função *printf()*.

```
char str[10]="Brasil";
for(int i=0;i<10;i++){
  if(str[i]!='\0'){
    printf("%c ", str[i]);
  }
}
```

As funções printf(), scanf(), puts(), gets() e outras funções que trabalham com *strings* já utilizam o caractere de finalização de *string* por padrão.

8.3.1. Funções para captura e exibição de *strings*

Para capturar e exibir *strings* podemos utilizar os já conhecidos printf() s scanf() ou ainda outras funções conforme abaixo:

- **printf() e scanf()**

Utilizando a máscara %s conseguimos capturar e exibir *strings*. Veja um exemplo:

```
#include <stdio.h>
int main(void) {
  char str[20];
  printf("Informe uma string:");
  scanf("%s", &str); //captura uma string
  printf("A string informada foi: %s", str); //Exibe a string
  return 0;
}
```

- **puts() e gets()**

Outras funções muito utilizadas são a puts (put string) que em inglês significa algo como “colocar uma string”, e, gets (get string) que significa “pegar” uma string. Na prática eles funcional como printf s scanf respectivamente, mas com algumas características diferentes. Por exemplo, não é possível usar máscaras com gets

O programa abaixo realiza exatamente a mesma coisa que o programa anterior, ou seja, captura e exibe uma string na tela.

```
#include <stdio.h>
int main(void) {
  char str[20] = "Brasil";
  printf("Informe uma string:");
  gets(str);
  puts(str);
  return 0;
}
```

8.3.2. Funções de manipulação de *strings*

Devido a complexidade de se trabalhar com *strings*, algumas funções foram criadas especialmente para elas. Por exemplo, podemos comparar duas variáveis de qualquer tipo (*char, float, int*, etc) através do operador '==' mas no caso de *strings* a operação envolve comparar letra a letra do início ao fim, então a biblioteca da linguagem C já dispõe de uma função que faz esse trabalho. Abaixo falaremos dela e de outras funções.

- **strcpy**

Sintaxe: strcpy(string_destino, string_origem);

Apenas na criação de uma string (vetor de caracteres) é possível atribuir valores com o sinal de atribuição "=". Exemplo:

```
char str[10]="Brasil";
```

No restante do código, como não é possível atribuir um valor diretamente para uma *string* na forma str="Brasil", então a função ***strcpy*** (*string copy*) faz este papel.

- **strcat**

Sintaxe: strcat(string_destino, string_origem);

Realiza a concatenação do conteúdo de uma variável a outra. Ambas devem ser strings.

- **strlen**

Sintaxe: variável_do_tipo_inteiro = strlen(string);

Determina e retorna o tamanho de uma *string*.

- **strcmp**

Sintaxe: variável_do_tipo_inteiro = strcmp(stringA, stringB);

Compara o conteúdo de duas *strings* pela ordem alfabética de seus caracteres e retorna os seguintes valores:

0 (zero): Se o conteúdo das *strings* são iguais
Inteiro < 0: Se o conteúdo da stringA é menor do que stringB
Inteiro > 0: Se o conteúdo da stringA é maior do que stringB

Agora veja um exemplo de programa que utiliza todas as funções citadas acima.

```c
#include <stdio.h>
#include <string.h>
#define TAM_STR 255

int main(void) {

  //Criação das variáveis
  char strA[TAM_STR];//String não inicializada
  char strB[TAM_STR];//String não inicializada
  char strC[TAM_STR];//String não inicializada
  //Captura de dados
  printf("Informe a string A:");
  gets(strA);
  printf("Informe a string C:");
  gets(strB);

  //Exibe os valores
  puts("\n\nValores informados:");
  printf("String A: %s\n", strA);
  printf("String B: %s\n", strB);
  //Função strcpy
  strcpy(strC, "Brasil");
  printf("String C: %s (atribuido via código)\n\n", strC);

  //Função strcmp
  int comparacao;
  comparacao = strcmp(strA,strB);
  if(comparacao==0){
    puts("Strings A e B são iguais!");
  }else if(comparacao<0){
    printf("A ordem alfabética das strings é: %s\n%s\n",strA,strB);
  }else{
    printf("A ordem alfabética das strings é: %s\n%s\n",strB,strA);
  }

  //Função strlen
  printf("\nA String A tem %d caracteres.", strlen(strA));
  printf("\nA String B tem %d caracteres.", strlen(strB));
  printf("\nA String C tem %d caracteres.", strlen(strC));
  printf("\nA String C é: ");
  for(int i=0; i<strlen(strC); i++){
    printf("*%c", strC[i]);
  }
  puts("*");

  //Função strcat
  strcat(strA,strB);
  puts("\nAs duas primeiras strings juntas são:");
  puts(strA);
  strcat(strA,strC);
  puts("\nAs 3 strings juntas são:");
  puts(strA);
  return 0;
}
```

8.3.3. Vetores de *strings*

Agora que já nos acostumamos a trabalhar com *strings* precisamos entender como utilizar listas ou um conjunto de várias *strings* ao mesmo tempo. Por exemplo, caso seja necessário armazenas uma lista de nomes de pessoas, como fazer utilizamos o que vimos até agora? Na prática isso é impossível, a não ser que seja criada uma variável (*string*) para cada nome na lista. Mas isso torna o trabalho pouco flexível. Precisamos criar um vetor de *strings* e para isso precisamos utilizar uma matriz de caracteres de duas dimensões. Veja:

índice/valor	0	1	2	3	4	5	6	7	8	9
0	'B'	'r'	'a'	's'	'l'	'\0'				
1	'B'	'l'	'u'	'm'	'e'	'n'	'a'	'u'	'\0'	
2	'J'	'o'	'a'	'ç'	'a'	'b'	'a'	'\0'		
3	'C'	'a'	'p'	'i'	'n'	'z'	'a'	'l'	'\0'	
4	'J'	'o'	'i'	'n'	'v'	'i'	'l'	'l'	'e'	'\0'

No exemplo acima temos uma matriz de 5 elementos na primeira dimensão e 10 elementos na segunda. Na prática isso significa que conseguimos armazenar 5 nomes com 10 caracteres cada. Lembre-se do '\0' ao final da *string*, portanto, se o tamanho é 10 então temos 9 caracteres úteis para utilizar em cada *string*.

Para declarar a lista acima usamos o seguinte comando:

```
char lista[5][10]; //criação da matriz
```

Já para interagir com os elementos precisamos informar a primeira dimensão como no código abaixo que irá exibir na tela "Blumenau" que é o nome de nossa queria cidade da *Oktoberfest*.

```
printf("%s", lista[1]);
```

Perceba que se desejamos acessar a letra 'm' da palavra Blumenau usamos o comando com "%c" ao invés de "%s" e com as duas dimensões da matriz:

```
printf("%c", lista[1][3]);
```

Abaixo temos o código para capturar uma lista de 5 nomes com até 255 caracteres cada um.

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

int main(void) {

  char lista[5][255]; //criação da matriz
```

```
  //captura de dados
  for(int i=0;i<5;i++){
    printf("String %d:", i+1);
    gets(lista[i]);
  }

  //impressão dos dados
  for(int i=0;i<5;i++){
    printf("\nString %d: %s", i+1, lista[i]);
  }

  return 0;
}
```

Todas as outras funções para manipulação de *strings* funcionam com matrizes de caracteres também. Fim do capítulo!

8.4. EXERCÍCIOS PROPOSTOS

Abaixo temos exercícios envolvendo vetores, ou seja, matrizes de apenas uma dimensão.

8.1) Faça um programa que cria um vetor com 3 elementos inteiros, lê os 3 números do teclado, armazena os números em um vetor e imprime o vetor na saída padrão.

8.2) Faça um programa que cria um vetor com 5 elementos inteiros, lê 5 números do teclado, armazena os números no vetor e imprime o vetor na ordem inversa.

8.3) Escreva um programa que lê 5 números inteiros. Após a captura dos 5 números mostrar os todos os 5 números na tela um ao lado do outro separados por espaços. O programa deve imprimir na linha abaixo o maior dos números com a seguinte frase: “O maior dos números acima é: <numero>”.

8.4) Escreva um programa que lê 10 números reais. O programa deve imprimir a média, o maior e o menor dos números.

8.5) Escreva um programa para ler as notas de 5 avaliações de um aluno (utilize vetores para armazenar), depois deve-se calcular e imprimir a média semestral. Crie uma função para validar as notas. Faça com que o algoritmo só aceite notas válidas (uma nota válida deve pertencer ao intervalo [0,10]). Cada nota deve ser validada separadamente. Deve ser impressa a mensagem "Nota inválida" caso a nota informada não pertença ao intervalo [0,10].

8.6) Reescreva o programa anterior para que no final seja impressa a mensagem “Novo cálculo (1=Sim 2=Não)” solicitando ao usuário que informe um código (1 ou 2) indicando se ele deseja ou não executar o programa novamente. Se for informado o código 1 deve ser repetida a execução de todo o programa para permitir um novo cálculo, caso

contrário ele deve ser encerrado. Valide a resposta do usuário para a pergunta “Novo cálculo (1=Sim 2=Não)” aceitando apenas os códigos 1 ou 2.

8.7) Faça um programa que leia um número indeterminado de idades (crie um vetor com 10 mil elementos). A última idade lida, que não entrará nos cálculos, deverá ser igual a zero, ou seja, se a idade informada for zero o programa finaliza e exibe os dados de quantas idades foram lidas. Deve-se também calcular e escrever a média de idade desse grupo de idades.

8.8) Faça um programa para ler a altura e o sexo (feminino ou masculino) de 10 pessoas. Ao final calcular e escrever na saída padrão:

a) A maior e a menor altura
b) A média de altura das mulheres
c) O número de homens

8.9) Faça um programa que toma as notas dos alunos (15 números fracionários) do usuário e guarda em um vetor. Depois disso, calcula e imprime na tela a média dos alunos, a nota mais alta, a nota mais baixa e o número de notas 7.0.

8.10) Faça um programa que ordene um vetor de dez números em ordem crescente. Ex.: inicialmente ele vale {2,78,5,-1,0,3,3,56,7,77} e é impresso na tela na sequência -1, 0, 2, 3, 3, 4, 5, 7, 56, 77. Sugestão: ache o menor valor do vetor e coloque na primeira posição; daí ache o segundo menor (basta procurar da segunda posição em diante) e coloque na segunda posição, e assim por diante. Este método é conhecido como inserção direta. Há vários outros algoritmos de ordenação (pesquise na internet).

Abaixo temos exercícios envolvendo matrizes com duas dimensões.

8.11) Faça um programa que lê uma matriz de 3 x 3 elementos usando um comando for, multiplica cada elemento por 5 e imprime o resultado.

8.12) Faça um programa que lê um vetor de 3 elementos e uma matriz de 3 x 3 elementos. Em seguida o programa deve fazer a multiplicação do vetor pelas colunas da matriz.

8.13) Faça um programa que apresente automaticamente uma matriz de dimensões 4x5, sendo que o valor de A(i, j) = 3i + j^2.

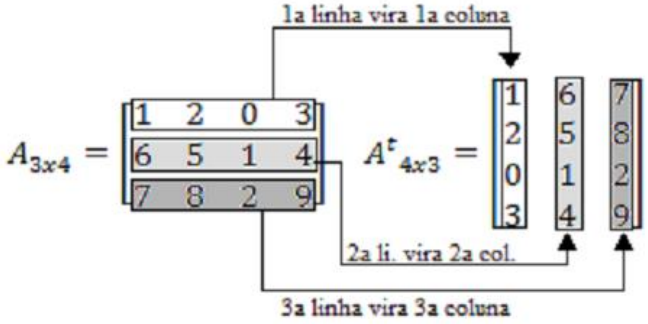
8.14) Faça um programa que receba duas matrizes de dimensões 3x3 e apresente a soma de seus elementos. Lembre-se que a adição de duas matrizes se dá pela soma de seus termos correspondentes. Esquematicamente, se A + B = C, então $c_{ij} = a_{ij} + b_{ij}$, ou seja, a soma de duas matrizes só é possível quando elas possuem a mesma ordem (número de linhas e colunas):

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

Onde:

$$\begin{cases} c_{11} = a_{11} + b_{11} \\ c_{12} = a_{12} + b_{12} \\ c_{21} = a_{21} + b_{21} \\ c_{22} = a_{22} + b_{22} \end{cases}$$

8.15) A matriz transposta de uma matriz A qualquer é denominada por A^t. Esta matriz nada mais é do que a matriz A com suas linhas transformadas em colunas e suas colunas transformadas em linha.
Observe:



Faça um programa que receba as dimensões da matriz, depois os valores e apresenta a matriz transposta.

8.16) Através de laços de repetição e decisão imprima uma matriz B = (b_{ij}) 5x5 tal que:

$$b_{ij} \begin{cases} -2 \; se \; i > j \\ 1 \; se \; i = j \\ 2 \; se \; i < j \end{cases}$$

Abaixo temos exercícios envolvendo strings que são vetores de caracteres ou ainda matrizes de caracteres que podem armazenar várias strings.

8.17) Faça um programa que lê duas palavras do teclado e diz se elas são iguais ou diferentes. O programa deve conferir e exibir um relatório informando se alguma das palavras digitadas é igual a “Brasil” ou “Santa Catarina”.

8.18) Faça um programa que lê uma frase do teclado e indica na tela quantas vezes a letra “s” aparece na frase.

8.19) Faça um programa que lê três palavras do teclado e imprime as três palavras na ordem inversa.

8.20) Faça um programa que criptografa uma mensagem informada pelo usuário (via scanf() ou gets()) e exibe a mensagem criptografada na tela. A criptografia deverá ser realizada alterando os caracteres através da soma de 1 inteiro referente ao código da tabela ACSII ou seja, a letra ‘a’ na mensagem será exibida como ‘b’, a letra ‘d’ como ‘e’ e assim por diante.

8.21) Escreva um programa em linguagem C para capturar o nome e idade de várias

pessoas. Ao final do primeiro cadastramento de nome e idade o programa deve solicitar ao usuário que seja informado ‘c’ para continuar ou ‘s’ para sair. Ao final do cadastramento deve ser exibida uma lista com os nomes ao lado das idades e na última linha a quantidade de nomes e a média das idades informadas.

8.22) Dada a matriz de duas dimensões de caracteres abaixo que foi definida pelo nome de variável cincoPalavras

	0	1	2	3	4	5	6
0	P	E	D	R	O		
1	M	A	R	I	A		
2	J	E	S	U	S		
3	I	F	C				
4	L	U	Z	E	R	N	A

O que será impresso em tela quando da execução da linha abaixo?

```
printf(“%c”, cincoPalavras[2][2]);
```

8.23) Qual a função do programa a seguir?

```
int main(){
    int z;
    float vetor_cubo[5], vetor[5];
    for(z=0; z<5; z++){
        printf("Digite um numero: ");
        scanf("%f",&vetor[z]);
        printf("\n");
        vetor_cubo[z]= pow(vetor[z],3);
    }
    for(z=0; z<5; z++)
        printf("%3.f\t", vetor_cubo[z]);
    printf("\n");
    return(0);
}
```

a) Ele preenche um vetor com 5 números e mostra outro vetor com os números em ordem crescente;
b) Ele preenche um vetor com 5 números e mostra outro vetor com o cubo dos números do primeiro vetor;
c) Ele preenche um vetor com 5 números e mostra outro vetor com o quadrado dos números do primeiro vetor;
d) Ele preenche um vetor com 5 números e mostra outro vetor com o cubo dos números do primeiro vetor e a soma deles;
e) Nenhuma das alternativas.

8.24) Marque a resposta que corresponde ao que o programa gera na saída padrão.

```
#include <stdio.h>
int  main () {
   int v [10] = {-1, -2, -3, -4, -5, -6, -7, -8, -9, -10 };
   int i, maior , s;
   maior = s = 0;
   for (i = 0; i < 10; i++) {
      s += v[i];
      if (v[i] > maior ) {
         maior = v[i];
      }
   }
   printf ("%d %d\n", maior , s);
   return 0;
}
```

Assinale a alternativa correta:

a) -1 -55
b) 0 -55
c) -10 -55
d) -55 -10
e) -55 0

9. Arquivos em disco

Até agora trabalhamos sempre com a memória principal do sistema, ou seja, a memória RAM (Random Access Memory) que é muito rápida mas tem a desvantagem de ser completamente apagada quando a máquina é desligada. Portanto, os dados até agora são voláteis, ou seja, não duram pra sempre. Para resolver este problema vamos aprender a trabalhar com arquivos em disco (memória secundária) que pode ser desde um HD (Hard Disk) tradicional até um SSD (Solid State Drive) ou um pendrive por exemplo.

Em resumo, arquivos tem a seguinte vantagem:

1. São permanentes e mantém os dados mesmo após a máquina ser desligada.
2. Permitem armazenar grande quantidade de dados, normalmente mais que a memória principal.
3. Pode ser acessado concorrentemente por vários programas ao mesmo tempo.

Os arquivos podem ser do tipo texto ou do tipo binário. Arquivos tipo texto normalmente possuem extensão “txt” (texto) ou “csv” (*comma-separated values*)[12] que são arquivos que podem ser abertos automaticamente por planilhas eletrônicas. Atualmente arquivos no formato JSON são muito utilizados pois permites estruturas dados complexos. JSON vem de *JavaScript Object Notation* que é um formato leve de troca de dados. É fácil para humanos ler e escrever e é fácil para as máquinas analisarem e manipularem. Já arquivos binários são específicos do sistema que os criaram então são menos indicados atualmente.

Para trabalhar com arquivos é preciso entender o funcionamento de um ponteiro em Linguagem C. Um ponteiro basicamente é uma variável inteira que possui um número de endereço da memória principal. Na prática já trabalhamos com ponteiros quando começamos a trabalhar com vetores, pois o nome do vetor (sem utilizar o índice) armazena o ponteiro para o início do vetor em memória principal.

9.1. ABERTURA E FECHAMENTO DE ARQUIVOS

Um arquivo é uma sequência de bytes e para manipulá-lo usaremos as funções auxiliares da Linguagem C para trabalhar com arquivos, sendo que a principal delas é a abertura do arquivo que irá retornar um ponteiro para o início da sequência de bytes do arquivo. Ao abrir o arquivo, se ele não existir um novo será criado na mesma pasta de execução do programa principal. Mas, se necessário, é possível definir um caminho

[12] Na computação, um arquivo de valores separados por vírgula (CSV) é um arquivo de texto delimitado que usa uma vírgula para separar valores. Um arquivo CSV armazena dados tabulares (números e texto) em texto simples. Cada linha do arquivo é um registro de dados. Cada registro consiste em um ou mais campos, separados por vírgulas. O uso da vírgula como um separador de campo é a origem do nome desse formato de arquivo.

relativo ou absoluto.

O primeiro passo é criar uma variável do tipo FILE que é um ponteiro para o arquivo que será aberto.

```
FILE *fp; //lembre-se do '*'
```

O segundo passo é abrir o arquivo mantendo fp como referência (ponteiro).

```
fp = fopen("teste.txt", "w");
```

Após a abertura, neste caso em modo ‘w’ que é escrita, podemos fazer a gravação de informações. Veremos como gravar e ler informações mais adiante.

Veja as possibilidades de abertura de arquivo na Linguagem C:

Modo	Significado
r	Abre o arquivo somente para leitura. O arquivo deve existir. (O *r* vem do inglês *read*, ler)
r+	Abre o arquivo para leitura e escrita. O arquivo deve existir.
w	Abre o arquivo somente para escrita no início do arquivo. Apagará o conteúdo do arquivo se ele já existir, criará um arquivo novo se não existir. (O *w* vem do inglês *write*, escrever)
w+	Abre o arquivo para escrita e leitura, apagando o conteúdo pré-existente.
a	Abre o arquivo para escrita no final do arquivo. Não apaga o conteúdo pré-existente. (O *a* vem do inglês *append*, adicionar)
a+	Abre o arquivo para escrita no final do arquivo e leitura.

Após abrir o arquivo e trabalhar com ele é muito importante fechar o mesmo arquivo, caso contrário seu programa em C poderá ser fechado mas o arquivo ficará aberto no sistema operacional alocando recursos desnecessários e impedindo que outro programa acesse o arquivo.

Para fechar o arquivo use a função *fclose()*:

```
fclose (fp);
```

Antes de finalizar esta primeira parte é importante lembrar que ao trabalhar com arquivos muitas coisas podem dar errado. O disco pode estar cheio e não aceitar a criação de um novo arquivo. O disco também pode falhar e retornar um erro na escrita de algo novo ou na leitura do arquivo. Também é possível que ocorram erros na hora de fechar o arquivo. É por isso que as funções para trabalho com arquivos retornam códigos para informar se esta tudo bem ou se um erro ocorreu.

A função *fopen()* retorna NULL se o arquivo não foi aberto corretamente. Já a função *fclose()* retorna um inteiro igual a zero se o arquivo foi fechado com sucesso ou algo diferente de zero se um erro ocorreu. Portanto, o código completo da abertura e

fechamento de um arquivo em disco segue abaixo:

```c
#include <stdio.h>
int main(void) {
  FILE *fp; //lembre-se do '*'
  fp = fopen("teste1.txt", "w"); //abre o arquivo em modo gravação
  if (fp == NULL) { //verifica se abriu arquivo com sucesso
    printf ("Erro ao abrir arquivo!\n");
    return 1; //finaliaa o programa retornando 1 ao sistema operacional
  }
  printf ("Arquivo criado com sucesso!\n");
  int status; //cria variável para armazenar o status do fechamento
  status = fclose (fp); //fecha arquivo
  if(status == 0){ //se status é igual a zero conseguiu fechar com sucesso
    puts("Arquivo fechado com sucesso!\n");
  } else {
    puts("Erro ao fechar arquivo!\n");
  }
  return 0;
}
```

9.2. ESCREVENDO EM ARQUIVOS

As três principais funções para imprimir na saída padrão que utilizamos até agora são *printf(), puts() e putchar()*. Essas mesmas três funções tem suas respectivas versões para impressão em arquivos. Veja:

Saída padrão	Arquivos	Explicação
putchar	**fputc**	Imprime apenas um caractere.
puts	**fputs**	Imprime uma *string* diretamente, sem nenhuma formatação.
printf	**fprintf**	Imprime uma *string* formatada.

Nas duas primeiras precisamos incluir um segundo parâmetro que é o ponteiro para o arquivo. Já para a *fprinf()* o ponteiro do arquivo é o primeiro argumento utilizado. Veja os exemplos de código:

```c
#include <stdio.h>

int main(void) {
  FILE *fp; //lembre-se do '*'
  fp = fopen("teste.txt", "w");
  if (fp == NULL) {
    printf ("Erro ao abrir arquivo!\n");
    return 1;
  }
```

```
  printf ("Arquivo criado com sucesso!\n");

  //Escrita no arquivo
  printf ("Iniciando a escrita.\n");
  fputc('A', fp);
  fputs("\nTeste fputs()...", fp);
  fprintf (fp, "\nTeste fprintf() número inteiro: %d", 42);
  fprintf (fp, "\nTeste fprintf() número real: %.2f", 1.1);
  printf ("Escrita finalizada.\n");

  int fechou;
  fechou = fclose (fp);
  if(fechou == 0){
    puts("Arquivo fechado com sucesso!\n");
  } else {
    puts("Erro ao fechar arquivo!\n");
  }
  return 0;
}
```

O resultado que pode ser conferido abrindo o arquivo gerado com um editor de texto simples será o seguinte:

```
A
Teste fputs()...
Teste fprintf() número inteiro: 42
Teste fprintf() número real: 1.10
```

Também é possível utilizar a função *fwrite()* que envolve conceitos de ponteiros e de vetor para escrita em arquivos. Esta função esta fora do escopo deste livro.

9.3. LENDO ARQUIVOS

Para ler de arquivos usamos funções análogas as da escrita. Lembre-se que o arquivo deve ser aberto em modo leitura e cada modo de abertura posiciona o ponteiro de leitura ou no início ou no final do arquivo. Veja as funções disponíveis:

Saída padrão	Arquivos	Explicação
getchar	**fgetc**	Recebe apenas um caractere.
gets	**fgets**	Lê uma *string* (geralmente uma linha inteira).
scanf	**fscanf**	Recebe uma *string* formatada.

Veja um exemplo para ler, linha a linha, um arquivo texto:

```c
#include <stdio.h>

int main(void) {
  FILE *fp; //lembre-se do '*'
  fp = fopen("teste.txt", "r");
  if (fp == NULL) {
    printf ("Erro ao abrir arquivo!\n");
    return 1;
  }
  printf ("Arquivo criado com sucesso!\n");

  //Leitura do arquivo
  printf ("Iniciando a leitura.\n");
  char temp[255]; // string temporária
  int num_linha=1;
  while(fgets (temp, 255, fp) != NULL){
    printf("Linha %d = ", num_linha++);
    printf(temp);
  }
  printf ("\nLeitura finalizada.\n");

  int fechou;
  fechou = fclose (fp);
  if(fechou == 0){
    puts("Arquivo fechado com sucesso!\n");
  } else {
    puts("Erro ao fechar arquivo!\n");
  }
  return 0;
}
```

Como você percebeu acima a função *fgets()* tem 3 argumentos sendo que o primeiro é a *string* que irá receber a linha lida. O segundo é o número máximo de caracteres que a função irá ler, contudo, a ideia é a leitura da linha encerrar antes do número máximo ao alcançar '\n'. O terceiro é o ponteiro para o arquivo.

Existem inúmeras maneiras de utilizar funções para leitura e escritas em arquivos. Com as explicações desenvolvidas e os exemplos mostrados neste capítulo esperamos que você consiga iniciar suas gravações e leituras de arquivos com sucesso! Não deixe de fazer os exercícios propostos no caderno de exercícios anexos ao livro.

Finalizamos o capítulo, e o livro, desejando bons estudos e muito sucesso em sua carreira profissional! A programação (*coding*), não há dúvida, é uma habilidade, ou como os americanos chamam: "skill", fundamental no mundo contemporâneo. Evolua sempre!

9.4. EXERCÍCIOS PROPOSTOS

9.1) Faça um programa que grava uma palavra em um arquivo chamado “palavra.txt”, a palavra deve ser digitada pelo operador.

9.2) Faça um programa que lê a palavra do arquivo do exercício anterior e a apresenta na tela.

9.3) Faça um programa que grava uma matriz de 3 x 3 números inteiros em um arquivo chamado “mat.txt”, os números devem ser digitados pelo operador.

9.4) Faça um programa que lê a matriz do arquivo do exercício anterior e a apresenta na tela.

9.5) Faça um programa utilizando o comando “for” que lê 10 números do teclado e armazene estes 10 números em um arquivo chamado “num.txt” na área de trabalho do computados.

9.6) Faça um programa que lê 20 palavras de um arquivo e verifica se a palavra “casa” está entre elas.

9.7) Faça um programa que lê 5 notas de um arquivo chamado “notas.txt”, calcula a média destas notas, imprime a média destas notas e salva a média no final deste mesmo arquivo.

9.8) Faça um programa utilizando o comando “switch-case” que pede para o operado escolher entre três opções: (S) para sair, (L) para ler a matriz e (G) para gravar a matriz. Se o operador escolher a opção S o programa é finalizado. Se o operador escolher a opção L o programa deve pedir ao operador qual o nome do arquivo e em seguida ler a matriz deste arquivo e apresentar a matriz na tela. E se o operador escolher a opção G o programa deve pedir para o operador digitar o nome do arquivo, o número de linhas e o número de colunas da matriz, e em seguida deve pedir para o operador digitar os números que compõem a matriz. A matriz deve ser salva no arquivo especificado. Não esqueça de incluir no arquivo o número de linhas e colunas da matriz para que a leitura dos dados seja possível.

9.9) Escreva um programa que leia todos os caracteres do arquivo “texto.txt” e que imprima por ordem decrescente as percentagens das ocorrências das letras encontradas. Os caracteres que não são letras devem ser ignorados. Exemplo:

```
Quantidade de caracteres no arquivo:
 a = 11.4%
 i = 11.1%
 e =  9.7%
 t =  9.3%
 ...
```

9.10) A empresa “Top Bussiness Inc.” é uma organização com mais de 2 mil funcionários que está tendo problemas de espaço em disco no seu servidor de arquivos. Para tentar resolver este problema, o Administrador de Rede precisa saber qual o espaço em disco ocupado pelas contas dos usuários, e identificar os usuários com maior espaço ocupado. Através de um aplicativo baixado da Internet, ele conseguiu gerar o seguinte arquivo, chamado “usuarios.txt“:

```
Ariovaldo;54253425
Cesar;65436248
Ricardo;32189784
João;3219684321
Maria;3219843218
```

O layout do arquivo demonstra que o primeiro campos (até o “;”) corresponde ao login do usuário e o segundo ao espaço em disco ocupado pelo seu diretório de arquivos. Agora crie um programa que gere um relatório, chamado “relatório.txt”, no seguinte formato:

```
Top Bussiness Inc.
------------------------------------------------


Nr.  Usuário        Espaço utilizado     % do uso
1    XXX            99,99 MB            99,99%
2    XXX            99,99 MB            99,99%
3    XXX            99,99 MB            99,99%
4    XXX            99,99 MB            99,99%
5    XXX            99,99 MB            99,99%


Espaço total ocupado: 999,99 MB
Espaço médio ocupado: 999,99 MB
```

9.11) Faça um programa para ler e exibir na tela os dados de um arquivo no seguinte formato (layout):

```
3
ZE; 8.5; 10.0
ANTONIO; 7.5; 8.5
SEBASTIAO; 5.0; 6.0
```

Nesta entrada, a primeira linha contém o número de alunos que serão inseridos. As linhas seguintes contêm os seguintes dados:

- nome do aluno com no máximo 50 caracteres;
- nota da primeira prova;
- nota da segunda prova

9.12) Escreva um programa que leia um arquivo no layout acima e imprima os nomes de todos os alunos que têm a média das duas notas menor que 7.0.

10. Operadores de bits

Existem operadores de bits para trabalhar especificamente com números binários. Isso é extremamente importante principalmente quando trabalhos em baixo nível (nível de bit) com portas lógicas ou com microcontroladores. Nestes equipamentos, é importante saber manipular números a nível de bit para poder, por exemplo, ligar e desligar individualmente cada bit.

Como exemplo temos um inteiro de 1 byte que resulta em 8 bits. Estes 8 bits, transmitidos para uma porta de um microcontrolador podem ser o comando para ligar e desligar 8 *leds* ou motores por exemplo. O microcontrolador irá receber o número e irá, conforme seus bits, ligar e desligar os motores e *leds*.

Por exemplo, o número 1 em decimal é igual a 0b00000001 em binário com 8 bits. Portanto, só o último bit estará ligado. Já o decimal 7 é igual a 0b00000111 em binário com 8 bits e, portanto, teríamos os 3 últimos bits ligados. Lembre-se que o prefixo “0b” (Zero mais a letra b) significa que o restante do número está em formato binário. Os operadores binários são:

Expressão Original	Expressão equivalente	Descrição
x=x>>k;	x>>=k;	Deslocamento de bits à direita
x=x<<k;	x<<=k;	Deslocamento de bits à esquerda
x=x&k;	x&=k;	E para bits (AND)
x=x|k;	x|=k;	OU para bits (OR)
x=x^k;	x^=k;	OU Exclusivo para bits (XOR)
x=~k	Não se aplica	NOT ou Complemento de 1

Veremos agora cada um destes operadores. Mas antes é importante destacar que convencionamos em chamar o bit mais a direita de bit “zero”, o segundo da direita para a esquerda de bit “1”, o terceiro “2” e assim por diante. Veja:

Índice do bit	7	6	5	4	3	2	1	0
Número binário	?	?	?	?	?	?	?	?

Esta convenção irá facilitar a compreensão do conteúdo e dos códigos daqui em diante. Por exemplo, o número 0b01010101 tem os bit 0, 2, 4 e 6 estão ligados e os bits 1, 3, 5 e 7 estão desligados.

10.1. OPERADOR DE DESLOCAMENTO À DIREITA >>

Realiza o deslocamento bit a bit à direita perdendo o(s) bit(s) mais a direita. O(s) novo(s) bit(s) à esquerda são sempre zeros. Como exemplo, temos que 0b00001000 com deslocamento de 1 bit à direita resultará em 0b00000100.

Deslocar um bit a direita faz o valor na base decimal de um inteiro ser dividido por 2. O último bit a direita que é perdido na operação é justamente a perda da divisão inteira.

```
unsigned char x;// 1 byte = 8 bits
x=9; // igual a 0b00001001
x=x>>1; // muda para 0b00000100
printf("%d", x); //imprime 4
```

Podemos substituir a expressão `x=x>>1;` por `x>>=1;` mantendo o resultado.

10.2. OPERADOR DE DESLOCAMENTO À ESQUERDA <<

O operador de deslocamento a esquerda funciona de forma análoga ao deslocamento a direita. Deslocar um bit a esquerda faz o valor na base decimal de um inteiro ser multiplicado por 2.

```
int main(void) {
  unsigned char x;// 1 byte = 8 bits
  x=5; // igual a 0b00000101
  x<<=1; // muda para 0b00001010
  printf("%d", x); //imprime 10
  return 0;

}
```

Uma operação muito comum no trabalho com microcontroladores é mandar um inteiro (8 bits) com apenas um dos bits ligados para acionar um comando. Por exemplo, para mandar ligar o bit 2 os comandos abaixo resolvem a questão:

```
unsigned char x;// 1 byte = 8 bits
x=1; // igual a 0b00000001
x=x<<2; // muda para 0b00000100
printf("%d", x); //imprime 4
```

Podemos substituir a expressão `x=x<<2;` por `x<<=2;` com o mesmo resultado.

10.3. OPERADOR E (AND) BINÁRIO

O operador binário AND, ou em português “E”, é muito utilizado para desligar um determinado bit.

Vamos supor que é preciso desligar o bit 5. Neste caso basta criar um inteiro com todos os bits ligados (ou seja, iguais a 1) e apenas o bit 5 desligado (igual a zero). A este inteiro auxiliar daremos o nome de máscara. Feito isso basta fazer a operação AND da variável alvo com a máscara. Abaixo temos um exemplo da operação AND com a máscara para desligar o bit 5 mantendo os outros bits intactos.

```
Variável alvo: 0b00110001
Máscara:       0b11011111
Resultado AND: 0b00010001
```

Perceba que apenas nos locais onde os dois bits estão ligados é que o bit será mantido ligado. Portanto, apenas o bit 5, destacado em vermelho, será alterado. Abaixo segue o código completo que realiza uma expressão como esta.

```
#include <stdio.h>

int main(void) {
  unsigned char x;// 1 byte = 8 bits
  x=49; // igual a 0b00110001
  unsigned char mascara;// 1 byte = 8 bits
  mascara=0b11011111; // Atribuição tendo “0b” como prefixo
  x=x&mascara; // muda para 0b00010001
  printf("%d", x); //imprime 17
  return 0;
}
```

10.4. OPERADOR OU (OR) BINÁRIO

Da mesma forma que é possível utilizar uma máscara para desligar um bit com o operador AND, é possível ligar um bit com o operador OR.

Para ligar o primeiro e o terceiro bit de uma variável alvo utilizamos o procedimento de criar uma máscara com o bit 0 e 2 ligados e depois realizamos a operação OR.

```
int main(void) {
  unsigned char x;// 1 byte = 8 bits
  x=0; // igual a 0b00000000
  unsigned char mascara;// 1 byte = 8 bits
  mascara=0b00000101; // Atribuição tendo “0b” como prefixo
  x=x|mascara; // muda para 0b00000101
  printf("%d", x); //imprime 5
  return 0;
}
```

Para facilidade na criação da máscara, utiliza-se o operador de deslocamento à esquerda, ficando da seguinte forma a ligação do bit 4:

```
int main(void) {
  unsigned char x;// 1 byte = 8 bits
  x=0; // igual a 0b00000000
  x=x|(1<<4); // muda para 0b00010000
  printf("%d", x); //imprime 16
```

```
  return 0;
}
```

10.5. OPERADOR NÃO EXCLUSIVO (XOR) BINÁRIO

O operador XOR é utilizado para inverter o sinal de um bit específico. Para isso é preciso criar uma máscara para utilizar na operação XOR. Todos os bits ligados da máscara irão ser invertidos na variável alvo. Veja um exemplo que inverte o bit 1 da variável alvo:

```
int main(void) {
  unsigned char x;// 1 byte = 8 bits
  x=49; // igual a 0b00110001
  x = x ^ (1<<1); // utiliza a máscara gerada por 1<<1 que é 0b00000010
  printf("%d", x); //imprime 51 que é equivalente a 0b00110011
  return 0;
}
```

Perceba acima que apenas o bit 1 foi invertido. Veja mais um exemplo:

```
int main(void) {
  unsigned char x;// 1 byte = 8 bits
  x=49; // igual a 0b00110001
  x = x ^ 0b00110001; // operação com o próprio número
  printf("%d", x); //imprime 0
  return 0;
}
```

Neste caso utilizamos exatamente o mesmo número como máscara, então apenas os bits ligados (1) foram invertidos para zero, o que resulta em todos os bits desligados e consequentemente o valor decimal zero.

10.6. OPERADOR NÃO (NOT) OU COMPLEMENTO DE 1

O operador NOT é utilizado para inverter o estado dos bits da variável alvo. Voltemos ao exemplo utilizado com o operador AND para desligar um bit:

```
int main(void) {
  unsigned char x;// 1 byte = 8 bits
  x=49; // igual a 0b00110001
  unsigned char mascara;// 1 byte = 8 bits
  mascara=0b11011111; // Atribuição tendo “0b” como prefixo
  x=x&mascara; // muda para 0b00010001
  printf("%d", x); //imprime 17
  return 0;
}
```

É possível criar a máscara de forma mais rápida utilizando o operador NOT, veja:

```
int main(void) {
  unsigned char x;// 1 byte = 8 bits
  x=49; // igual a 0b00110001
  x = x & ~(1<<5); // utiliza a máscara 0b11011111
  printf("%d", x); //imprime 17
  return 0;
}
```

Veja que neste caso foi criada a máscara `1<<5` que é igual a 0b00100000 e depois invertidos todos os seus bits para 0b11011111 com a ajuda do operador NOT (~).

10.7. TESTANDO O VALOR DE UM BIT

É muito comum fazermos a leitura de um determinado bit em alguma porta de um microcontrolador. Para isso, leremos a porta através de uma variável do tipo inteiro, neste caso continuamos utilizando o tipo *char* por ser um inteiro de 8 bits na linguagem C.

Para ler o valor do bit 2 temos:

```
PORTA:         0b????????
Máscara:       0b00000100
Resultado AND: 0b00000?00
```

Portanto, estamos utilizando o comando AND para verificar o valor de um bit específico de uma maneira muito simples, veja:

```
int main(void) {
  unsigned char PORTA;// 1 byte = 8 bits
  PORTA=0b00001111;
  if(PORTA & (1<<2)){ //(1<<2) gera a máscara 0b00000100
    puts("Bit ligado!");
  }
  return 0;
}
```

Uma outra maneira de fazer a leitura de um determinado bit será mostrada abaixo nas macros.

10.8. MACROS PARA MANIPULAÇÃO DE BITS

Todo mundo que trabalha com microcontroladores ou de alguma forma opera no baixo nível dos bits precisa definir, inverter e limpar bits individuais em algum momento, e tudo isso precisa ser feito sem alterar os outros bits. Essas operações, conforme já mostradas acima, requerem comandos simples, mas muitas vezes difíceis de lembrar, que incluem comandos OR, AND, XOR e NOT.

Para facilitar este trabalho, podemos definir (usando o comando `#define`) macros que são uma espécie de “mini funções” para essas operações. Vamos tratar das macros mais comuns abaixo. Fizemos as explicações nos comentários do código para que você possa copiar e colar no seu próprio código. Também incluímos a função *imprimeBin()* que imprime um número decima em formato binário.

```
#include <stdio.h>

/* Esta macro altera o bit y (indexado com base zero) de x para '1', através
da geração de uma máscara com '1' no bit indicado e fazendo a operação OR do
x com a máscara. */
#define SET(x,y) (x |= (1 << y))

/* Esta macro altera o bit y (indexado com base zero) da variável x para '0'
gerando uma máscara com '0' na posição y e 1's no restante e então faz a
operação AND. */
#define CLEAR(x,y) (x &= ~(1<< y))

/* Esta macro inverte o valor do bit de x indicado na posição y, usando a
operação XOR com uma máscara onde o bit na posição y é '1'e todos os outros
são '0'. */
#define TOGGLE(x,y) (x ^= (1<<y))

/* Esta macro retorna '1' se o valor do bit na posição y é '1', caso
contrário retorna '0'. A operação é feita através do operador AND com uma
máscara contendo '1' na posição indicada e '0' no restante. */
#define READ(x,y) ((0u == (x & (1<<y)))?0u:1u)

void imprimeBin(unsigned char x){
  unsigned char restos[8];
  printf("0b");
  for(int i=0;i<8;i++){
    restos[i]=x%2;
    x/=2;
  }
  for(int i=7;i>=0;i--){
    printf("%d", restos[i]);
  }
  puts("");
}

int main(void) {

  unsigned char x;// 1 byte = 8 bits
  x=0b00001111;
  puts("Valor inicial:");
  imprimeBin(x); //imprime 0b00001111
```

```
  puts("Ligando o bit 6 temos:");
  SET(x,6);
  imprimeBin(x); //imprime 0b00001111

  puts("Desligando o bit 2 temos:");
  CLEAR(x,2);
  imprimeBin(x); //imprime 0b00001111

  puts("Invertendo o bit 0 temos:");
  TOGGLE(x,0);
  imprimeBin(x); //imprime 0b00001111

  //Lê o bit 0
  printf("\nO bit 0 é: %d\n", READ(x,0));

  //Lê todos os bits
  puts("\nLeitura de todos os bits de x:");
  for(int i=0;i<8;i++){
    printf("O bit %d é: %d\n", i, READ(x,i));
  }

  return 0;
}
```

A saída do código será:

```
Valor inicial:
0b00001111
Ligando o bit 6 temos:
0b01001111
Desligando o bit 2 temos:
0b01001011
Invertendo o bit 0 temos:
0b01001010

O bit 0 é: 0

Leitura de todos os bits de x:
O bit 0 é: 0
O bit 1 é: 1
O bit 2 é: 0
O bit 3 é: 1
O bit 4 é: 0
O bit 5 é: 0
O bit 6 é: 1
O bit 7 é: 0
```

10.9. EXERCÍCIOS PROPOSTOS

10.1) Escreva uma função que receba um valor em decimal, converta para binário e dentro da própria função exiba na tela o resultado. A função terá tipo de retorno void.

10.2) Faça uma função chamada “int bin2dec(int b1, int b2, int b3, int b4)” que receba um número binário de 4 bits e converta para decimal. Cada bit deverá ser um argumento inteiro da função. Exemplo de utilização: “int decimal = bin2dec(1,0,1,1);”

10.3) Faça a função “qtdeDeBitsParesLigados(int n)” que retorna o número de bits pares que estão ligados. Exemplo:

```
main(){
   qtdeDeBitsParesLigados(73);
   //se 73 = 0100 1001, então qtdeDeBitsParesLigados retorna 2
}
```

10.4) Escreva uma função criptografa(int n) que recebe um inteiro n com 8 bits (índices: 7,6,5,4,3,2,1,0) e que retorna esse inteiro embaralhando esses bits para a seguinte sequência (7,5,3,1,6,4,2,0). Exemplo:

```
main(){
    criptografa(73); // se 73 = 0100 1001
    //então criptografa(73) == 0010 1001 == 41
}
```

10.5) Faça uma função pisca() que ora liga todos os bits pares e ora liga todos os bits ímpares num intervalo de 2 segundo entre as mudanças, imitando o efeito de um pisca-pisca de natal. Apresente os valores em binários de como ficaria a saída. A palavra de trabalho deverá ter 8 bits. O valor de milissegundos é usado para fazer uma espera entre a alternância, e para isso use a função “Sleep(<int milissegundos>)” da biblioteca windows.h. A saída terá as linhas abaixo sendo impressas uma a uma a cada 2 segundos:

```
Saída:
01010101
10101010
01010101
10101010
01010101
10101010
```

10.6) Faça uma função “piscaUmIndoEVoltando(int milissegundos)” que faz os leds pares serem ligados sequencialmente, um de cada vez e de forma crescente, e em seguida, os leds ímpares serem ligados de forma decrescente. Apresente na tela o resultado em formato binário. A palavra de trabalho deverá ter 8 bits. Cada linha será impressa apenas depois de x milissegundos (parâmetro da função). A sequência gerada deverá ser:

```
Saída:
01000000
01010000
01010100
01010101
01010111
01011111
01111111
11111111
```

10.7) Faça um programa que leia um byte do teclado e a seguir zere os bits 3 e 4, e inverta os bits 0 e 7. O resultado deverá ser mostrado em hexadecimal na tela.

10.8) Construa funções que realizem as seguintes operações aplicadas aos valores 0b10101, 0b11111, 0b11100:

a) Mostre o valor do primeiro bit da direita
b) Mostre o terceiro bit da direita para a esquerda
c) Mostre o primeiro bit da esquerda
d) Mostre o terceiro bit da esquerda para a direita
e) Mostre os dois primeiros bits da esquerda para a direita
f) Mostre os bits pares
g) Altere o primeiro bit da direita para 0
h) Altere o primeiro bit da esquerda para 0
i) Inverta todos os bits (o que for 0 deverá ser 1 e o que for 1 deverá ser 0)
j) Mostre apenas 4 bits da direita, removendo os outros

10.9) Faça uma função “consecutivos(int x)” que retorne 1, caso o inteiro c possua 2 ou mais bits consecutivos ligados.

10.10) Crie a função “rotacionaParaDireita(int x, int qtde)” que retorna x com n rotações à direita sem perder os bits.

10.11) Faça a função “cripto(int x, int rotacoes)” que recebe um inteiro x com 8 bits com os bits tendo índices base zero (7,6,5,4,3,2,1,0) e que retorna esse inteiro rotacionado para a esquerda, sem perder bits, com o número de rotações do parâmetros “rotações”. Faça posteriormente a função “descripto(int x, int rotações)” que faz a rotação inversa e

recupera as informações.

11. Apêndice A: Filmes que você precisa assistir

A história da Microsoft e da Apple foi representada no filme: Piratas do Vale do Silício. Sobre o filme, Bill Gates em entrevista Reddit disse: “Minha representação foi razoavelmente precisa…”. Portanto, você precisa ver esse filme!

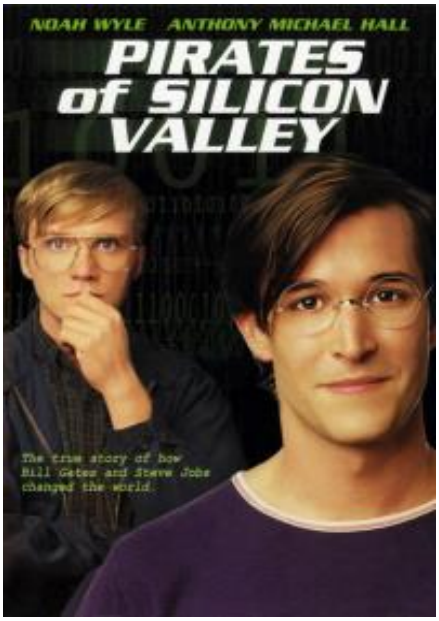


Figura 5 Filme Piratas do Vale do Silício a verdadeira história da Apple e da Microsoft.

Para mais filmes sobre Jobs, Gates e a história da Microsoft e da Apple veja esse artigo do TechTudo: “Conheça três filmes que explicam a história...”, disponível em:

<www.techtudo.com.br/artigos/noticia/2013/02/conheca-tres-filmes-que-explicam-historia-da-microsoft-e-da-apple.html>.

Já que estamos falando de história, que tal assistir a entrevista histórica juntando Bill Gates da Microsoft e Steve Jobs da Apple? A entrevista ocorreu em 2007 quando os dois estavam no auge. Steve Jobs faleceu em 2011. Assista Steve Jobs e Bill Gates juntos. Disponível em: <youtu.be/hh1rVsYtQMA>.

Se quiser aproveitar veja Bill Gates novamente, dessa vez falando de assuntos gerais com o Warren Buffett, segundo homem mais rico do mundo (afinal Bill é o primeiro). Bill Gates e Buffet - Conselhos aos jovens. Disponível em <youtu.be/VQ5y45cPQzA>.

Reforço também a indicação do filme já mencionado no início do livro. O filme “The Imitation Game” que no Brasil foi chamado de “O Jogo da Imitação” conta a história de Alan Turing, o pai da computação. Até hoje os computadores utilizam a arquitetura básica projetada por ele.

E por fim, é claro, não deixe de assistir Matrix com Keanu Reeves que, afinal, é o melhor filme de todos os tempos! E não esqueça que o grande arquiteto está de olho em você.

Figura 6 Filme “The Imitation Game”.