Programação de Computadores I

Joacir Giaretta
jgiarett@ucs.br

Introdução

O uso de algoritmos é quase tão antigo quanto a matemática. Com o passar do tempo, entretanto, ele foi bastante esquecido pela matemática. Com o advento das máquinas de calcular e mais tarde os computadores, o uso de algoritmos ressurgiu com grande vigor, como uma forma de indicar o caminho para a solução dos mais variados problemas.

Algoritmo não é a solução do problema, pois, se assim fosse, cada problema teria um único algoritmo. Algoritmo é um caminho para a solução de um problema, e em geral, os caminhos que levam a uma solução são muitos.

Ao longo dos anos surgiram muitas formas de representar os algoritmos, algumas utilizando linguagens semelhantes às linguagens de programação e outras utilizando formas gráficas, como os fluxogramas.

Dentre as formas de representação usadas, deu-se uma acentuada preferência por formas estruturadas, cuja principal vantagem é a de facilitar a legibilidade e compreensão dos algoritmos.

O aprendizado de algoritmos não se consegue a não ser através de muitos exercícios.

Não se aprende algoritmos:

- Copiando algoritmos
- Estudando algoritmos

Só se aprende algoritmos:

- Construindo algoritmos
- Testando algoritmos

Conceitos Básicos

Essa seção visa conceituar constante, variável, operação, expressão e atribuição. Considere-se a fórmula matemática simples do cálculo do volume de uma esfera:

$$V = \frac{4}{3} \pi R^3$$

onde se encontram:

1- valores que podem ser classificados como:
    a) valores constantes, invariantes em todas as aplicações da fórmula, no caso, os valores 4, 3 e π, aos quais denomina-se **constantes**.
    b) valores a serem substituídos na fórmula, em cada aplicação. A representação destes valores, usualmente, é feita através de letras que recebem o nome de **variáveis** e tornam a fórmula genérica, possível de ser aplicada para resolver uma certa classe de problemas.
2- Operações a serem feitas sobre determinados operandos (valores), para a obtenção da solução do problema.

Na fórmula do volume da esfera, estão representadas as operações de divisão, multiplicação e potenciação, feitas sobre os operandos, 4, 3, π e R, numa sequência pré-determinada, mas nem sempre rigorosa, devido às leis matemáticas.

Sob este aspecto, uma fórmula matemática é simplesmente uma descrição de um conjunto de **ações** que devem ser executadas sobre um conjunto de **objetos**, sendo estes, constantes ou variáveis.

Existe uma outra relação importante entre objeto e operação, no sentido de que, certas operações somente podem ser executadas sobre determinados objetos. Por exemplo:

- a raiz quadrada de um número qualquer somente pode ser feita se o número for real e positivo
- o cálculo do fatorial de um número somente pode ser feito se o número for inteiro e positivo
- a determinação do arco seno trigonométrico somente pode ser feita se o valor estiver no intervalo [-1 , 1].

Nos exemplos apresentados, pode-se notar dois tipos de limitações:

1- limitação de classe de valores: inteiros, reais, lógicos, literais
2- limitações de intervalo, pois algumas operações não atuam sobre todo conjunto, mas somente sobre parte dele.

Os tipos de dados (valores) existentes podem, então, ser inteiros, reais, caracteres (ou literais) e lógicos (ou booleanos). Esses tipos básicos são bem definidos e podem ser combinados de modo a formar tipos de dados estruturados, incluindo vetores, matrizes, registros e arquivos.

Uma **constante** é, então, um objeto invariante em cada execução de uma expressão, enquanto **variável** é um objeto que representa um valor que pode ser alterado a cada execução da expressão em que estiver inserida.

As variáveis podem ser visualizadas como receptáculos de valores. Cada receptáculo é identificado através de um **nome** e somente pode receber valores de um determinado tipo.

Uma vez conceituados constante e variável pode-se analisar as combinações deles na construção de expressões. Expressões, no sentido matemático, são representações

simbólicas de sequências de operações a serem feitas sobre determinados operandos, visando a obtenção de um resultado.

O conjunto de operandos e operações que compõem uma expressão determina o tipo da expressão e define o tipo de resultado a ser obtido. As expressões mais frequentemente usadas na solução de problemas enquadram-se em uma das seguintes categorias:

- expressões aritméticas: produzem como resultado um número
- expressões literais: produzem como resultado um texto
- expressões lógicas: produzem como resultado um valor logico.

Para cada uma das categorias acima, pode ser definido um conjunto de operações possíveis. Por exemplo, as expressões aritméticas podem utilizar as operações de adição, subtração, multiplicação, divisão, entre outras. As expressões literais podem utilizar a operação de concatenação, unindo dois conjuntos de caracteres.

Nesse primeiro instante, irão ser estudadas as expressões aritméticas. Na resolução de uma expressão aritmética deve-se seguir uma ordem pré-definida dependente dos operadores que aparecem nesta expressão. A prioridade dos operadores básicos é:

1- potenciação, radiciação e operações unárias
2- multiplicação e divisão
3- soma e subtração
4- parênteses podem alterar esta ordem
5- segue-se da esquerda para a direita em caso de indeterminação (mais de uma operação com a mesma prioridade)

A última operação realizada em uma expressão se chama **atribuição de valor**, que é o ato de armazenar ou escrever um valor em uma variável. A atribuição compreende uma ação de substituição do conteúdo de uma variável cujo efeito é: após a atribuição, a variável passa a representar um novo valor, sendo "perdido" o seu valor anterior.

Em algoritmos, existem várias maneiras de representar uma atribuição de valor, como:

```
variavel = valor
variavel := valor
variavel <- valor
```

Será utilizada a primeira forma para representar uma atribuição. Veja um exemplo de atribuição utilizando variáveis:

A variável `A` tem valor `2` e `B` valor `4` e se quer intercambiar os valores entre eles. Nesse caso, não é possível escrever simplesmente:

```
A = B
B = A
```

Cujo resultado seria `A` com valor `4` e `B` com valor `4` também. Nesse caso tem-se que utilizar uma variável auxiliar, assim:

```
C = A
A = B
B = C
```

Algoritmos Puramente Sequenciais

Essa seção visa conceituar algoritmo e caracterizar algoritmos puramente sequenciais, além de apresentar a formalização da linguagem algorítmica.

Em termos conceituais, algoritmo é **um conjunto finito de regras, bem definidas, para a solução de um problema em um tempo finito**. Considerando o seguinte problema: dados três valores não nulos a, b, c, determinar a média aritmética. As tarefas a serem executadas para a solução deste problema podem ser descritas assim:

1- obter os valores de a, b, c
2- calcular a média aritmética pela fórmula: média aritmética = $\frac{a + b + c}{3}$
3- comunicar os resultados obtidos: média aritmética
4- terminar

A sequência de tarefas apresentada, especificada passo a passo, é um algoritmo, pois é um conjunto finito de regras que podem ser consideradas bem definidas e cuja execução produz a solução do problema proposto, após um tempo finito.

No exemplo das médias foi utilizada uma linguagem algorítmica informal, onde não existem regras rígidas na construção do algoritmo. Para os algoritmos desenvolvidos a partir de agora, será utilizada uma linguagem algorítmica estruturada, baseada na linguagem C.

Antes de tudo, é importante conhecer a estrutura de um programa em linguagem C.

Estrutura de um programa em C

Um programa em C é constituído de:

- um cabeçalho contendo as diretivas de compilador onde se definem o valor de constantes simbólicas, declaração de variáveis globais, inclusão de bibliotecas, declaração de rotinas, etc.
- um bloco de instruções principal (função principal) e outros blocos de funções.
- documentação do programa: comentários.

Comentários

Em C, comentários podem ser escritos em qualquer lugar do programa para facilitar a sua posterior interpretação. Para que o comentário seja identificado como tal, ele deve ter um `/*` antes e um `*/` depois, no caso de querer comentar um bloco de instruções, ou `//` no caso de querer comentar linhas individuais. Exemplo:

```
/* esta é uma
linha de
comentário em C */
```

ou:

```
// esta linha está comentada
```

Dica: No Eclipse CDT, quando se quer fazer comentários, seja de uma ou várias linhas, basta utilizar a combinação `Ctrl` + `/`.

Diretivas de Compilação

Em C, existem comandos que são processados durante a **compilação** do programa. Estes comandos são genericamente chamados de **diretivas de compilação**. Uma delas é a informação de quais **bibliotecas** devem ser anexadas ao programa executável. A diretiva `#include` diz ao compilador para incluir na compilação do programa outros arquivos. Geralmente estes arquivos contém bibliotecas de funções ou rotinas do usuário. Exemplos de bibliotecas do C:

`stdlib.h`	Funções básicas do C (obrigatória)
`stdio.h`	Funções de entrada e saída em disco
`ctype.h`	Funções que tratam de caracteres
`string.h`	Funções de tratamento de strings
`mem.h`	Funções que manuseiam a memória
`math.h`	Funções matemáticas
`alloc.h`	Funções de alocação dinâmica de memória
`graphics.h`	Funções de manuseio do vídeo no modo gráfico
`conio.h`	Funções de manuseio do vídeo no modo texto
`time.h`	Funções de manuseio de data e hora

O exemplo a seguir apenas mostra no vídeo o texto `Alo Mundo!!!` e aguarda até uma tecla ser digitada. Todo programa em C consiste em uma ou mais funções. A única função que necessariamente precisa estar presente é a denominada `main()`, que é a primeira função a ser chamada quando a execução do programa começa.

```
/* Programa simples para mostrar Alo Mundo na tela */

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

void main()
{
   printf("Alo Mundo!!!");
   getchar(); // em IDEs que executam no prompt de comando, é
necessário para visualizar o resultado
}
```

Percebe-se que a função `main()` está entre `{}`, indicando o início e o fim da função. O `printf` é responsável por mostrar o string `Alo Mundo!!!` na tela, sendo que todo string deve estar entre aspas, e o `getchar()` aguarda até que uma tecla seja pressionada. É importante frisar que o C diferencia caracteres minúsculos de maiúsculos, e que praticamente toda a programação é feita em minúsculo. O `void` antes do `main` significa que a função não retorna valor, o que será discutido adiante.

Estrutura de um algoritmo e funções de entrada e saída de dados

O algoritmo em si se constitui basicamente em:

- entrada de dados
- processamento
- saída de dados

Para a saída de dados será utilizada a instrução:

Instrução: printf()

Biblioteca: `stdio.h`

Exemplo:

```
   printf("Ola Mundo!!! \n");
   printf("linha 1 \nlinha 2");
}
```

O resultado será o a seguir apresentado:

```
Ola Mundo!!!
linha 1
linha 2
```

Observe que, na primeira instrução, a saída é exatamente igual ao string (nome dado a um texto em programação, normalmente entre aspas). Já na segunda instrução a impressão se deu em duas linhas. Isto se deve ao `\n` que representa a sequência de escape para nova linha.

É possível reservar espaço para o **valor** de alguma variável usando **especificadores de formato**. Um especificador de formato marca o **lugar** e o **formato** de impressão das variáveis contidas na **lista variáveis**. Deve haver um especificador de formato para cada variável a ser impressa. Todos os especificadores de formato começam com um `%`.

Observe, no exemplo abaixo, as instruções de saída formatada e os respectivos resultados. Admita que `idade` seja uma variável inteira (`int`) contendo o valor `29` e que `tot`, `din` e `troco` sejam variáveis reais (`float`) cujos valores são, respectivamente, `12.3`, `15.0`. e `2.7`.

Exemplo:

```
printf("Tenho %d anos de vida",idade);
printf("Total: %.2f \nDinheiro: %.2f \nTroco: %.2f", tot, din, troco);
```

Saída:

```
Tenho 29 anos de vida
Total: 12.30
```

```
Dinheiro: 15.00
Troco: 2.70
```

Depois do sinal `%`, segue o especificador de formato que representa o tipo de dado:

Especificador	Tipo de dado
`d`	inteiro decimal (int)
`u`	inteiro decimal sem sinal (unsigned int)
`hd`	inteiro decimal curto (short)
`hu`	inteiro decimal curto (unsigned short)
`ld`	inteiro decimal longo (long)
`lu`	inteiro decimal longo (unsigned long)
`f`	ponto flutuante (float)
`lf`	ponto flutuante de precisão dupla (double)
`o`	inteiro octal
`x`	inteiro hexadecimal
`c`	caracter simples
`s`	string

No caso dos tipos de ponto flutuante, é possível determinar o número de casas decimais colocando o `.2` (ou o número de casas decimais pretendido) entre o `%` e o especificador. Por exemplo: `%.2f`.

Para a entrada de dados será utilizada a instrução:

Instrução: scanf()

Biblioteca: `stdio.h`

O uso da função `scanf()` é semelhante ao da função `printf()`. A função lê da entrada padrão (em geral, teclado) um valor cujo tipo de dado é especificado pelos mesmos especificadores de formato do `printf()`. A armazenagem se dá no endereço da variável (por isso que deve ser utilizado o `&`). **Todas variáveis devem ser precedidos pelo caracter `&`, menos as variáveis do tipo string, que não devem ser precedidos pelo `&`.**

Sempre que se utiliza o `scanf()`, é necessária a utilização do `fflush(stdin);`, evitando que "sujeira" fique para a próxima leitura.

Exemplo:

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

void main()
{
   int idade;
   char nome[30];
```

```
   printf("Digite sua idade: ");
   scanf("%d",&idade);
   fflush(stdin);  // se não utilizar o fflush, poderá ter problemas

   printf("Digite seu nome: ");
   scanf("%s",nome);    // strings não utilizar '&' na leitura
   fflush(stdin);

   printf("%s, você tem %d anos. \n", nome, idade);
   getchar();
}
```

É necessário incluir a biblioteca onde essas instruções estão armazenadas, por isso se deve ter o `#include <stdio.h>` no início do programa.

Tipos de dados

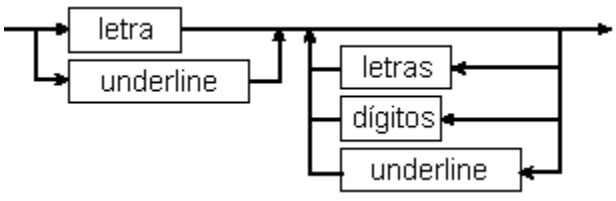
Como já foi comentado, o operador das atribuições será o `=`, mas as expressões aritméticas têm que estar dispostas todas na mesma linha, usando os operadores comumente utilizados em computadores. No exemplo da média aritmética, a expressão ficariam assim:

$$\frac{a + b + c}{3} \longrightarrow$$ `ma = (a + b + c ) / 3`

Os tipos de dados utilizados em variáveis são:

- `inteiro`: variáveis que armazenam valores inteiros
- `real`: variáveis que armazenam valores reais
- `caracter`: variáveis que armazenam um ou um conjunto de caracteres (string)
- `logico`: variáveis que armazenam verdadeiro ou falso (no caso de C, não há esse tipo, apenas em C++. No lugar dele será utilizada uma variável inteira, inserindo `0` para falso e `1` para verdadeiro.

Cada variável vai ter um nome identificador que deve ser iniciado por letras ou o caracter underline (_). O resto é formado por letras, dígitos e caracter underline. Não é permitida a utilização de espaços para a formação do identificador e nem acentuação. Exemplo de identificadores válidos: valortotal, soma_item1.



A linguagem C trata vários tipos de dados padrão (ou fundamentais). A declaração de variáveis deve se basear em um destes tipos. Os tipos de dados são usados para definir tamanho em bytes que uma variável vai ocupara na memória e qual o intervalo de valores que poderá armazenar.

Tipos Inteiros

São tipos numéricos exatos, sem casas decimais.

Tipo	Tamanho em Bytes	Valor Mínimo	Valor Máximo
`unsigned short`	2	0	65.355
`short`	2	-32.768	32.767
`unsigned int*`	2 ou 4	ver short e long	ver short e long
`int*`	2 ou 4	ver short e long	ver short e long
`unsigned long`	4	0	4.294.967.295
`long`	4	-2.147.483.648	2.147.483.647

* O tamanho de tipo de dado `int` depende no tamanho do tamanho da "palavra" do sistema operacional. Assim, no MS-DOS possui 16 bits, enquanto em sistemas de 32 bits (como Windows 9x/2000/NT) possui 32 bits (4 bytes). Em resumo, ele pode ser do mesmo tamanho que um `short`, ou que um `long`, dependendo da situação.

Tipos Reais

São tipos numéricos com casas decimais.

Tipo	Tamanho em Bytes	Valor Mínimo	Valor Máximo
`float`	4	3.4E-38	3.4E+38
`double`	8	1.7E-308	1.7E+308
`long double`	10	3.4E-4932	1.1E+4932

Tipos Caracter

São tipos que tratam de caracteres.

Tipo	Tamanho em Bytes	Valor Mínimo	Valor Máximo
`unsigned char`	1	0	255
`char`	1	-128	127

Um string é um conjunto ordenado de caracteres (texto) que pode ser armazenado sob forma de um vetor (esta estrutura de dados será vista em seções posteriores). Por enquanto, basta saber como declarar e armazenar um conjunto de caracteres em uma variável. **Sintaxe:** Para declarar uma variável para receber um conjunto de caracteres, deve-se identificar o tamanho desejado do string. Isso se faz através da declaração de um vetor do tipo `char`, assim:

```
char nome[40];
```

IMPORTANTE: Variáveis caracter representam uma única letra, dígito ou símbolo, e deve ser colocado entre **apóstrofes** (exemplo: `'a' 'b' 'X' '&'`). Já strings sempre devem ser expressas entre aspas (`"João"`).

Para que se possa usar uma variável em um programa, é necessário fazer uma declaração de variável antes. A declaração de variáveis simplesmente informa ao

processador quais são os nomes utilizados para armazenar dados variáveis e quais são os tipos usados. Deste modo o processador pode alocar (reservar) o espaço necessário na memória para a manipulação destas variáveis. É possível declarar mais de uma variável ao mesmo tempo, basta separá-las por vírgulas. A sintaxe para declaração de variáveis:

```
tipo variavel_1 [, variavel_2, ...] ;
```

Onde `tipo` é o tipo de dado e `variavel_1` é o nome da variável a ser declarada. Se houver mais de uma variável, seus nomes são separados por vírgulas. Exemplos:

```
int i;
int x,y,z;
char letra;
float nota_1,nota_2,media;
double num;
unsigned short idade;
```

A declaração de variáveis é feita, em geral, **dentro** de uma função. Por exemplo, a função principal `main()`. Deste modo se diz que está se fazendo uma declaração de variáveis **locais**. Variáveis locais podem ser referenciadas apenas dentro da função dentro da qual foi declarada, neste caso a função `main()`. **Exemplo:** Observe o uso da declaração de variáveis no trecho de programa abaixo:

```
void main()
{
   float raio, area;     /* declaracao de variáveis */
   raio = 2.5;
   area = 3.14;
}
```

Uma das operações fundamentais na programação de computadores é a atribuição de valor a uma variável. No C, o operador de atribuição é o `=`, como já foi visto em exemplos anteriores. Exemplo de declaração de variáveis locais e globais e atribuição de valor:

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

void main()
{
   double var_d;
   long var_l;
   int var_i1, var_i2, var_i3;
   float var_f = 5.6;
   char var_c = 'a';

   var_l = 50000;
   var_d = 100.5675;
   var_l = var_d;         // var_l = 100
   var_i2 = var_c;        // var_i2 = 97
   var_i1 = var_i2 = var_i3 = 0;
   getchar();      // aguarda até uma tecla seja pressionada
}
```

Todas as variáveis devem ser declaradas antes de serem utilizadas. Observa-se que é possível realizar a atribuição de valor na própria declaração da variável. O valor atribuído pode ser uma constante, ou uma outra variável.

Adicionalmente, não é possível realizar uma operação de atribuição à uma variável `string`, a não ser na sua declaração. A forma de se fazer isso é a partir de funções, vistas posteriormente.

Essa construção é possível:

```
char nome[40] = "João da Silva";
```

Essa construção não é possível:

```
   char nome[40];
   nome = "João da Silva";
```

É muito importante que se tenha cuidado com a sintaxe do algoritmo, ou seja, a grafia tem que estar rigorosamente dentro das especificações.

Algoritmo das médias em C

A partir do que foi visto, o algoritmo das médias fica assim:

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

void main()
{
   float a,b,c,ma;

   printf("Digite primeira nota: ");
   scanf("%f",&a);
   fflush(stdin);
   printf("Digite segunda nota: ");
   scanf("%f",&b);
   fflush(stdin);
   printf("Digite terceira nota: ");
   scanf("%f",&c);
   fflush(stdin);

   ma = (a + b + c ) / 3;

   printf("Media Aritmetica: %.2f", ma);
   getchar();
}
```

Percebe-se, nos exemplos, um pequeno avanço para a direita em alguns momentos. Isso se chama **endentação** do algoritmo/programa e, apesar de não influenciar na sua execução, é muito utilizada para sua organização e entendimento. Mais adiante, quando os algoritmos tomarem dimensões maiores e for necessário mais do que um nível de endentação, se notará mais claramente a sua importância.

DICA: Para fazer a endentação automática no Eclipse CDT, utilize a combinação de teclas `Ctrl` + `Shift` + `f`.

Exercícios

Os primeiros exercícios serão feitos na IDE (Integrated Development Environment – Ambiente de Desenvolvimento Integrado) WebAlgo (que se encontra no Acervo da Disciplina), que é uma ferramenta desenvolvida por professores e alunos da UCS. Após algumas aulas, iremos utilizar IDEs profissionais, como o Ecliplse CDT e o Code::Blocks. Adicionalmente, no Acervo da Disciplina está a lista de problemas, onde parte dela será resolvida na disciplina e, portanto, descrita nessa apostila. Esses mesmos problemas são carregados no WebAlgo.

- *S00000050 - Faça um algoritmo que leia um valor N, representando o lado de um quadrado, e calcule e escreva a área do quadrado.*

- *S00000200 - Faça um algoritmo que leia dois valores reais v1 e v2 e calcule e escreva a área do triângulo que tem base igual a v1 e altura igual a v2. Dica: A área de um triângulo é dada pela expressão: (base x altura)/2.*

- *Para a próxima aula, fazer a TDE 1.*

Operadores de Atribuição Aritmética

É comum em programação ter que alterar o valor de uma variável realizando alguma operação aritmética com ela, como `a = a + 1` ou `valor = valor * 5`. Embora seja perfeitamente possível escrever estas instruções, foi desenvolvido na linguagem C uma forma **otimizada**, através do uso de operadores ditos **operadores de atribuição aritmética**, que são `+=`, `-=`, `*=`, `/=`, `%=`. Deste modo, as instruções acima podem ser reescritas assim: `a += 1` e `valor *= 5`, respectivamente. A sintaxe da atribuição aritmética é a seguinte:

```
var += exp;
var -= exp;
var *= exp;
var /= exp;
var %= exp;
```

Onde `var` é o identificador da variável e `exp` é uma expressão válida. Estas instruções são equivalentes (mas não iguais) às seguintes:

```
var = var + exp;
var = var - exp;
var = var * exp;
var = var / exp;
var = var % exp;
```

Exemplo: Observe as atribuições aritméticas abaixo e suas instruções equivalentes:

Atribuição aritmética	Instrução equivalente
`i += 1;`	`i = i + 1;`
`j -= val;`	`j = j - val;`
`num *= 1 + k;`	`num = num * (1 + k);`
`troco /= 10;`	`troco = troco / 10;`
`resto %= 2;`	`resto = resto % 2;`

Operadores Incrementais

Na linguagem C existem instruções muito comuns chamadas de **incremento** e **decremento**. Uma operação de incremento **adiciona** uma unidade ao conteúdo de uma variável. Uma operação de decremento **subtrai** uma unidade do conteúdo de uma variável. Isso é realizado através de operadores específicos para realizar as operações de incremento (`++`) e decremento (`--`). Eles são genericamente chamados de **operadores incrementais**. **Sintaxe:**

Instrução	Instrução equivalente
`++var`	`var = var + 1`
`var++`	`var = var + 1`
`--var`	`var = var - 1`
`var--`	`var = var – 1`

Observe que existem duas sintaxes possíveis para os operadores: pode-se colocar o operador **à esquerda** ou **à direita** da variável. Nos dois casos o valor da variável será incrementado (ou decrementado) de uma unidade. Porém, se o operador for colocado **à esquerda** da variável, o valor da variável será incrementado (ou decrementado) **antes** que a variável seja usada em alguma outra operação. Caso o operador seja colocado **à direita** da variável, o valor da variável será incrementado (ou decrementado) **depois** que a variável for usada em alguma outra operação. **Exemplo:** Observe o fragmento de código abaixo e note o valor que as variáveis recebem **após** a execução da instrução:

```
int a, b, c, i = 3;      /* a: ?   b: ?   c: ?   i: 3 */
a = i++;                 /* a: 3   b: ?   c: ?   i: 4 */
b = ++i;                 /* a: 3   b: 5   c: ?   i: 5 */
c = --i;                 /* a: 3   b: 5   c: 4   i: 4 */
```

Os operadores incrementais são bastante usados para o controle de laços de repetição, que serão vistos em seções posteriores. É importante que se conheça exatamente o efeito sutil da colocação do operador, pois isto pode enganar o programador inexperiente.

Exercícios

- *S00000300 - Faça um algoritmo que leia 3 valores a, b e c, coeficientes de uma equação de segundo grau, e calcule e escreva a soma das raízes da equação. Dica: As raízes de uma equação podem ser calculadas pela fórmula de Baskhara.*

- *S00000900 - Faça um algoritmo que lê 3 valores, lados de um triângulo, e calcule e escreva a área do triângulo formado. Dica: A área de um triângulo de lados l1, l2 e l3 pode ser calculada pela expressão Área=raiz(S*(S-*

l1)(S-l2)*(S-l3)), onde S é o semi-perímetro, ou seja, a metade da soma dos lados.*

- *Exercício adicional: Considerando que o aumento dos funcionários é de 80% do INPC e mais um percentual de produtividade discutido com a empresa, escrever um algoritmo que lê o número do funcionário, seu salário atual, o índice INPC e o índice de produtividade conquistado e mostra o número do funcionário, seu aumento e o valor de seu novo salário.*

- *Para a próxima aula, fazer a TDE 2.*

Algoritmos com Seleção (Desvio Condicional)

Essa seção visa conceituar algoritmos com seleção, ou desvio condicional, e expressões lógicas, bem como explicitar as instruções existentes na linguagem algorítmica.

Existem algoritmos com situações resolvidas através de passos cuja execução é subordinada a uma condição. Tais algoritmos são denominados **algoritmos com seleção**. A condição aqui referenciada é denominada **expressão lógica** ou **expressão booleana**. A álgebra booleana é a que regulamenta as expressões lógicas, onde não existem outros estados possíveis além do verdadeiro e falso.

As expressões lógicas são construídas baseadas em dois tipos de operadores, os **operadores relacionais**, responsáveis pelas proposições realizadas e os **operadores lógicos**, responsáveis pela ligação de duas ou mais proposições.

Operadores Relacionais

Operador	Operação
`==`	Igual
`!=`	Diferente
`<`	Menor
`>`	Maior
`<=`	Menor ou igual
`>=`	Maior ou igual

Operadores Lógicos

Operador	Operação
`!`	“não” booleano
`&&`	“e” booleano
`||`	“ou” booleano

Desvio Condicional Simples

A instrução a ser utilizada é o `se...então...senão`, que em C é `if...else`:

Exemplo:

```
if (a > 2)
   printf("O valor é maior que 2");
```

Ou utilizando os operadores lógicos:

```
if (a > 0 && a < 2)
   printf("O valor é maior que 0 e menor que 2");
```

Quando existem instruções a serem executadas quando a expressão for falsa, deve-se utilizar o senão (`else`), assim:

```
if (a > 2)
   printf("O valor é maior que 2");
else
   printf("O valor é menor ou igual a 2");
```

E quando é necessário ter mais que uma instrução, tanto no `if` quanto no `else`, deve se criar um bloco de instruções, utilizando o `{}`, assim:

```
if (a > 2)
{
   printf("O valor é maior que 2. O novo valor é 4");
   a = 4;
}
else
{
   printf("O valor é menor ou igual a 2. O novo valor é 5");
   a = 5;
}
```

Atualmente, muitos programadores C deixam o código mais compacto, deixando as chaves nas mesmas linhas do `if` e do `else`, assim:

```
if (a > 2){
   printf("O valor é maior que 2. O novo valor é 4");
   a = 4;
} else {
   printf("O valor é menor ou igual a 2. O novo valor é 5");
   a = 5;
}
```

Na utilização dos operadores lógicos, os resultados possíveis são:

```
if (a > 0  &&  b < 0) ...
      V  | V |  V
      V  | F |  F
      F  | F |  V
      F  | F |  F

if (a > 0  ||  b < 0) ...
      V  | V |  V
      V  | V |  F
      F  | V |  V
      F  | F |  F
```

Pode-se ter expressões muito maiores, cujo resultado final vai ser verdadeiro ou falso, dependendo dos valores existentes nas variáveis envolvidas na expressão inteira, como:

```
if !((a > b && b > c) || (c > d && d > f)) ...
```

Exercícios

- *C00000040 - Faça um algoritmo que leia um valor e escreva: "Par", se o valor for par e "Ímpar", se o valor for ímpar.*

- *C00000060 - Faça um algoritmo que leia dois valores e, através de uma comparação, escreva o maior deles. Considere que os dois valores são diferentes.*

- *C00000100 - Faça um algoritmo que leia 3 valores v1, v2 e v3, e escreva-os em ordem crescente.*

- *C00000200 - Faça um algoritmo que leia 3 valores v1, v2 e v3 e coloque-os em ordem crescente, de forma que v1 contenha o menor, v2 contenha o elemento do meio (nem o maior, nem o menor), e v3 contenha o maior. Escreva os valores ordenados.*

- *C00000250 - Escreva um algoritmo que leia os valores das quatro provas de um aluno e escreva a média aritmética considerando apenas as três melhores notas. Por exemplo, se o valores lidos foram 9, 9.5, 7, e 8, a média será (9 + 9.5 + 8)/3 (a prova de nota 7 é descartada). Dica: Não esqueça de considerar a possibilidade de ocorrerem notas iguais.*

- *C00000300 - Faça um algoritmo que leia 3 valores a, b e c, coeficientes de uma equação de segundo grau, e verifique se a equação tem raízes reais. Se a equação tiver raízes reais, calcule e escreva as raízes da equação (em ordem crescente). Se não tiver, escreva "A equação não possui raízes reais". Dica: As raízes de uma equação podem ser calculadas pela fórmula de Baskhara. Uma equação não possui raízes se reais se $b^2-4ac < 0$*

- *C00000350 - Faça um algoritmo que leia 3 valores a, b e c, lados de um triângulo, e verifique o tipo de triângulo formado escrevendo: "Equlátero" - se o triângulo é equilátero (os três lados são iguais); "Isóceles" - se o triângulo é isósceles (dois lados iguais e um diferente); "Escaleno" - escaleno (3 lados diferentes).*

- *Exercício adicional: Escreva um algoritmo que leia o número do vendedor de uma empresa, seu salário fixo e o total de vendas por ele efetuadas. Cada vendedor recebe um salário fixo, mais uma comissão proporcional às vendas por ele efetuadas. A comissão é de 3% sobre o total de vendas até 1000 reais e 5% sobre o que ultrapassar esse valor. Mostrar a comissão e o salário total calculados.*

- *Para a próxima aula, fazer a TDE 3.*

Desvio Condicional Aninhado

Existem situações nas quais é necessário verificar condições de teste sucessivas, onde uma ação será executada caso um conjunto anterior de ações seja satisfeito. Podemos usar para resolver esse tipo de problema uma estrutura denominada Desvio Condicional Aninhado, que nada mais é do que o encadeamento de estruturas de decisão compostas em um algoritmo. Também chamamos a esse tipo de estrutura de Desvio Condicional Encadeado, por este motivo. **Sintaxe**:

```
if(condição 1)
{
  bloco 1;
}
else if(condição 2)
{
  bloco 2;
}
else if(condição N)
{
  bloco N;
}
else
{
  bloco P
}
```

Se a `condição 1` for **verdadeira,** o `bloco 1` é executado. Caso contrário, a `condição 2` é avaliada. Se a `condição 2` for **verdadeira,** o `bloco 2` é executado. Caso contrário, a `condição 3` é avaliada, e assim sucessivamente. Se nenhuma condição for **verdadeira**, o `bloco P` é executado. Observa-se que apenas um dos blocos é executado.

Exemplo: No trecho a seguir, uma determinada ação é executada se o valor de `num` for positivo, negativo ou nulo.

```
if(num > 0)
  a = b;
else if(num < 0)
  a = b + 1;
else
  a = b - 1;
```

Estrutura de Controle `switch...case`

A estrutura `switch...case` é uma estrutura de decisão que permite a execução de um bloco de instruções a partir de pontos diferentes, conforme o resultado de uma expressão de controle (que pode ser inteira ou caracter). O resultado desta expressão é comparado ao valor de cada um dos rótulos, e as instruções são executadas a partir desde rótulo. **Sintaxe**:

```
switch(expressão)
{
   case rótulo_1: bloco 1; break;
   case rótulo_2: bloco 2; break;
   ...
   case rótulo_n: bloco n; break;
```

```
    [default: bloco d]
}
```

onde `expressão` é uma expressão inteira ou caracter, `rótulo_1`, `rótulo_2`, `rótulo_n` e `rótulo_d` são constantes inteiras ou caracter e `bloco 1`, `bloco 2`, `bloco n` e `bloco d` são blocos de instruções.

O valor de expressão é avaliado e o fluxo lógico será desviado para o bloco cujo `rótulo` é igual ao resultado da expressão e todas as instruções **abaixo** deste rótulo serão executadas. Caso o resultado da expressão for diferente de todos os valores dos rótulos, então `bloco d` é executado. Os rótulos devem ser expressões constantes inteiras ou caracter **diferentes** entre si. O rótulo `default` é opcional.

Esta estrutura é particularmente útil quando se tem um bloco de instruções que se deve executar em ordem, porém se pode começar em pontos diferentes.

Exemplo: O trecho a seguir ilustra o uso da instrução `switch` em um menu de seleção. Neste exemplo, o programa iniciará o processo de usinagem de uma peça em um ponto qualquer, dependendo do valor lido.

```
int selecao;
printf("Digite estagio de usinagem:");
scanf("%d",&selecao);
fflush(stdin);
switch(selecao)
{
   case 1:  printf("Será realizado um desbaste grosso");
            break;
   case 2:  printf("Será realizado um desbaste fino");
            break;
   case 3:  printf("Será realizado o acabamento");
            break;
   case 4:  printf("Será realizado o polimento");
            break;
   default: printf("Estágio inexistente");
}
```

Percebe-se a existência da instrução `break` ao final de cada bloco de instruções. Isso é necessário, pois o `switch...case` não utiliza chaves para delimitação do início/fim de cada bloco.

Exercícios

- *C00000600 - Faça um algoritmo que leia três notas de um aluno e escreva sua média harmônica. Se o aluno obteve média abaixo de 6.0, E SOMENTE NESSE CASO, leia uma quarta nota (da prova de recuperação) e substitua a menor das três notas pela nota da recuperação e recalcule a média harmônica. Escreva a média harmônica final e o conceito obtido (0, se média harmônica (MH) < 6.0; 1 se 6.0 <= MH<7.0; 2 se 7.0 <= MH < 8.0; 3 se 8.0 <= MH < 9.0; 4 se MH>=9.0).*

- *C00000700 - As tarifas de um estacionamento são definidas assim: A primeira e a segunda hora custam 5 reais cada. A terceira e a quarta hora*

custam 2 reais cada. A partir da quinta hora, cada hora custa 1 real cada. Assim, se um carro ficar 5 horas no estacionamento, o motorista pagará 15 reais (5+5+2+2+1). Faça um algoritmo que leia dois valores He e Hs, respectivamente a hora de entrada e saída no estacionamento (horas inteiras, sem minutos), e escreva o valor a ser pago. Considere que o usuário deve retirar seu carro antes da meia-noite, ou seja, ele não pode entrar em um dia e sair no dia seguinte.

- *C00000750 - Faça um algoritmo que leia, para duas barras de ouro, o seu peso e seu valor. O algoritmo deve ler também o limite de peso de uma mochila, e verificar e escrever que barra (s) devem ir na mochila de modo a maximizar o valor dentro dela, sem exceder seu limite de peso. O algoritmo deve escrever as mensagens adequadas, como segue:*
    - *Nenhuma das barras puder ser colocada na mochila sem exceder o limite de peso*
    - *Apenas a barra 1 puder ir na mochila*
    - *Apenas a barra 2 puder ir na mochila*
    - *Ambas as barras puderem ir na mochila simultaneamente*
- *Exercício utilizando switch...case: Escrever um algoritmo que lê um conjunto de três valores reais qualquer a, b, c. Após, leia uma variável inteira "opcao" que é a escolha do usuário para mostrar os valores conforme o indicado a seguir:*
    - *Se opcao = 1 mostrar os 3 valores a, b, c em ordem crescente.*
    - *Se opcao = 2 mostrar os 3 valores a, b, c em ordem decrescente.*
    - *Se opcao = 3 mostrar os 3 valores de forma que o maior valor fique entre os outros dois.*
- *C00000800 - Faça um algoritmo que leia 4(quatro) valores e escreva os 3 (três) maiores em ordem decrescente. Considere que podem ocorrer valores iguais.*
- *C00000866 - Faça um algoritmo que leia a data de nascimento de uma pessoa e a data atual (cada data com dia, mês e ano). O algoritmo deve escrever 1 se a pessoa é maior de idade, e 0 se ela é menor de idade (maioridade: 18 anos).*
- *C00001000 - Faça um algoritmo que leia para um trabalhador o valor que ganha por hora, a hora de entrada e a hora de saída (valores inteiros, sem minutos) e calcule quanto ele ganhou pelo turno. Considere que ele entra e sai no mesmo dia, e que as horas a partir das 20:00 valem 20% a mais (adicional noturno).*
- *C00001100 - Faça um algoritmo que leia para um trabalhador o valor que ganha por hora, a hora de entrada e a hora de saída (valores inteiros, sem*

minutos) e calcule quanto ele ganhou pelo turno. Considere que ele entra e sai no mesmo dia, e que as horas antes das 6:00 da manhã e a partir das 20:00 valem 20% a mais (adicional noturno).

- *C00001250 - Faça um algoritmo que leia para um trabalhador o valor que ganha por hora, a hora de entrada e a hora de saída (valores inteiros, sem minutos) e calcule quanto ele ganhou pelo turno. Considere que ele pode entrar em um dia e sair no dia seguinte, e que se ele permanecer mais do que 8 horas, as duas horas a partir da nona hora valem 20% a mais, e as horas a partir da décima primeira hora valem 50% a mais (horas extras).*

Funções

Funções, também chamadas de **rotinas**, ou **sub-programas**, são a essência da programação estruturada. Funções são segmentos de programa que executam uma determinada tarefa específica. Já foram vistas funções nas seções anteriores, como `sqrt()`, `pow()`, `abs()`.

O programador pode escrever suas próprias funções. São as chamadas **funções de usuário** ou rotinas de usuário. Deste modo pode-se segmentar um programa grande em vários programas menores. Esta segmentação é chamada de **modularização de código** e permite que cada segmento seja escrito, testado e revisado individualmente, sem alterar o funcionamento do programa como um todo.

Estrutura das funções de usuário

A estrutura de uma função de usuário é muito semelhante a estrutura dos programas que escrevemos até agora. Uma função de usuário constitui-se de um **bloco de instruções** que definem os procedimentos efetuados pela função, um **nome** pelo qual a chamamos, uma **lista de argumentos** ou **lista de parâmetros** passados à função e o tipo de dado de **retorno**. Chamamos este conjunto de elementos de **definição da função**. Sintaxe:

```
tipo_de_retorno nome_da_função(tipo_1 arg_1, tipo_2 arg_2, ...)
{
  [bloco de instruções da função]
}
```

Exemplo: o código mostrado a seguir é uma função definida pelo usuário para calcular a média aritmética de dois números reais:

```
float media(float a, float b)
{
   float med;
   med = (a + b) / 2.0;
   return(med);
}
```

No exemplo acima, foi definida uma função chamada `media` que recebe dois argumentos tipo `float`: `a` e `b`. A média destes dois valores é calculada e armazenada na variável `med`, declarada internamente. Quando uma variável é declarada internamente, ela é chamada de **variável local** e seu escopo, ou sua visibilidade se restringe àquela

função. A função retorna, para o local que a chamou, um valor também do tipo `float`: o valor da variável `med`. Este retorno de valor é feito pela função `return()`. Depois de definida, a função pode ser usada dentro de uma outra função, ou até mesmo do `main()`. Diz-se que está sendo feita uma **chamada** à função.

Importante: Na maioria dos compiladores C, as funções devem, ou ser desenvolvidas antes do `main()` ou da função que a invoque, ou senão é obrigatória a declaração da função antes de sua chamada, ou seja, não é possível invocar uma função que está descrita após a sua invocação. As declarações vão depois dos `#include`. A declaração da função `float media(float a, float b)` ficaria assim:

```
float media(float, float);
```

Exemplo: No exemplo a seguir, é chamada a função `media()` dentro da função principal:

```
void main()
{
   float num_1, num_2, medarit;
   puts("Digite dois números: ");
   scanf("%f %f", &num_1, &num_2);
   fflush(stdin);
   medarit = media(num_1, num_2);   /* chamada a função  */
   printf("\nA media destes números e´ %f", medarit);
}
```

Percebe-se que a função `main()` é agora declarada com retorno `void`. O `void` indica que a função não tem retorno algum. Importante: Não é possível retornar mais do que um valor de uma função. Quanto aos parâmetros, não existe limite quanto ao número e, quando não houver nenhum parâmetro, simplesmente colocam-se apenas os parênteses. Exemplo: `int calculo()`.

Exercícios

- *Criar quatro funções:* `soma`, `subtracao`, `multiplicacao` *e* `divisao` *que devem receber como parâmetro dois valores* `float`, *fazer a operação correspondente e retornar o valor calculado (*`float`*). Após, na função* `main()`, *ler um valor* `float v1`, *um operador* `char op`, *que pode ser* `+`, `-`, `*`, `/`, *e um segundo valor* `float v2`. *As variáveis devem ser locais do* `main()`. *Através do* `switch`, *testar o operador e, conforme o caso, chamar uma das quatro funções criadas:* `soma`, `subtracao`, `divisao`, `multiplicacao`, *mostrando o resultado na tela.*

- *Fazer duas funções:*

  1) *Função para converter uma temperatura de Celsius para Fahrenheit. Deverá receber um parâmetro float e retornar um float, que é a temperatura convertida.*

     Protótipo: `float cels_fahr(float temp)`
     Exemplo: `cels_fahr(15)` deverá retornar `59`

2) *Função para converter uma temperatura de Fahrenheit para Celsius. Deverá receber um parâmetro float e retornar um float, que é a temperatura convertida.*

Protótipo: `float fahr_cels(float temp)`
Exemplo: `fahr_cels(59) deverá retornar 15`

Fórmulas:

```
Graus Celsius = Graus Fahrenheit – 32
                        1.8

Graus Fahrenheit = 1.8 x Graus Celsius + 32
```

Após, solicite a digitação de um `float`*, que é a temperatura em celsius. Converta para fahrenheit passando o valor por parâmetro para o* `cels_fahr` *e mostrar o retorno. Após, esse valor convertido para fehrenheit deve ser passado por parâmetro para o* `fahr_cels` *e mostrar o retorno, que deverá ser igual ao valor digitado inicialmente.*

- *Fazer duas funções:*

1. *Função para consistência de uma data. Deverá receber três parâmetros inteiros e retornar* `0` *se a data passada como argumento estiver errada e* `1` *se ok.*

   Protótipo: `int consiste_data(int dia, int mes, int ano)`
   Exemplo: `consiste_data(30,2,90)` deverá retornar `0` pois é uma data inválida.

2. *Função que receba, por parâmetro, uma data particionada em dia, mês e ano. A função deverá retornar o número do dia no ano. Dica: utilizar um vetor contendo os últimos dias de cada mês. Considere que ano bissexto é aquele divisível por 4 e que abril, junho, setembro e novembro tem 30 dias, fevereiro tem 28 (29 em ano bissexto) e todos os outros meses tem 31 dias.*

   Protótipo: `int data_nrodia(int dia, int mes, int ano)`
   Exemplo: `data_nrodia(31,12,92)` deverá retornar `366`.

Após, no `main()` *do programa, solicitar a digitação da data (três variáveis inteiras) e passar por parâmetro parra o* `consiste_data` *e, conforme a data for válida ou não, mostrar a mensagem adequada. Se a data for válida, após a mensagem de data válida mostrada, passar os valores digitados por parâmetro para o* `data_i` *e mostrar o retorno na tela.*

- *O dono de uma loja de venda de materiais de construção solicitou uma pequena aplicação que calculasse e mostrasse o número de galões de massa, fundo ou tinta que são necessários em uma determinada obra. Apesar de ser um valor estimado, é importante para o cliente saber aproximadamente quanto material deve ser adquirido. Alguns fabricantes informam uma fórmula básica para descobrir esse dado:*

$$\text{Número de galões} = \frac{\text{metragem quadrada} \quad X \quad \text{número de demãos}}{\text{rendimento por galão informado pelo fabricante}}$$

A partir da fórmula, fazer uma função que receba parâmetros para efetuar esse cálculo e retorne o número de galões necessários.

Protótipo:
```
float calc_tinta(float metragem, int demaos, float rendimento)
```

Exemplo: `calc_tinta(250,2,23)` *deverá retornar* `21.74`.

Após, no `main()` *do programa, fazer a leitura da metragem quadrada, do número de demãos e do rendimento por galão. Passar por parâmetro esses valores e mostrar na tela o resultado.*

Modelo de saída para a tela:

```
Metragem quadrada: 250
Número de demãos: 2
Rendimento por galão: 23

São necessários 21.74 galões de fundo/massa/tinta para essa
metragem.
```

Algoritmos com Repetição (ou Interativos)

Essa seção visa conceituar algoritmos com repetição, apresentar as instruções para manipulação desse tipo de algoritmo e variáveis do tipo contador e do tipo acumulador.

Todo algoritmo que possui um ou mais de seus passos repetidos um determinado número de vezes denomina-se **algoritmo com repetição**. São basicamente três as instruções existentes, vistas nas próximas seções.

Estrutura de Repetição `while`

A estrutura de repetição condicional `while` segue a sintaxe:

```
while(condição)
{
   bloco de instruções
}
```

onde `condição` é uma expressão lógica ou numérica e `bloco de instruções` é um conjunto de instruções.

Esta estrutura faz com que a condição seja avaliada em primeiro lugar. Se a condição for **verdadeira** o bloco é executado uma vez e a condição é avaliada novamente. Caso a condição seja **falsa** a repetição é terminada sem a execução do bloco

Em muitos casos, há a necessidade da utilização de **variáveis contadoras** dentro de um bloco de repetição. As variáveis contadoras recebem um valor inicial (geralmente 0) fora do bloco e é incrementada dentro do bloco (geralmente em 1).

Em outros casos, há a necessidade de utilizar **variáveis acumuladoras**, onde o valor inicial geralmente é zero e é incrementado de um valor variável, e não de uma constante.

Exemplo com variável contadora: No trecho a seguir, uma variável inicializada com zero é incrementada em um a partir de uma variável contadora e mostrada enquanto for menor ou igual a cinco.

```
cont = 0;
while(cont <= 5)
{
   cont = cont + 1;
   printf("%d\n", cont);
}
```

Exemplo com variável acumuladora: No trecho a seguir, uma variável inicializada com zero é incrementada em um a partir de uma variável contadora e somada (acumulada) em uma variável acumuladora enquanto for menor ou igual a cinco. Tal variável é mostrada após a repetição.

```
cont = 0;
soma = 0;
while(cont <= 5)
{
   cont = cont + 1;
   soma = soma + cont;
}
printf("A soma dos valores de 1 a 5 é %d", soma);
```

Exercícios

- *I00000100 (modificada) - Faça um algoritmo que escreva os números em ordem decrescente, de 20 a 1.*

- *I00000200 - Faça um algoritmo que escreva todos os números pares entre 1 e 50.*

- *I00000300 - Faça um algoritmo que escreva todos os números entre 1 e 200 que são múltiplos de 11.*

- ***Exercício adicional:*** *Faça um algoritmo que leia 10 números em uma única variável v. A cada leitura, acumular o valor em uma variável acumuladora e mostrar ao final.*

- *Para a próxima aula, fazer a TDE 5.*

Estrutura de Repetição `do...while`

A estrutura de repetição condicional `do...while` segue a sintaxe:

```
do{

    bloco de instruções
}while(condição);
```

onde `condição` é uma expressão lógica e `bloco de instruções` é um conjunto de instruções.

Esta estrutura faz com que o bloco de instruções seja executado pelo menos uma vez. Após a execução do bloco, a condição é avaliada. Se a condição for **verdadeira** o bloco é executado outra vez, caso contrário a repetição é terminada.

Exemplo 1: No trecho a seguir, uma variável inicializada com zero é incrementada em um e mostrada enquanto for menor que cinco.

```
cont = 0;
do{
   cont = cont + 1;
   printf("%d\n", cont);
}while(cont < 5);
```

Exemplo 2: No trecho a seguir, a leitura de um número é feita dentro de um laço de repetição condicional. A leitura é repetida caso o número lido seja negativo.

```
do{
   printf("Digite um número positivo: ");
   scanf("%f",&num);
   fflush(stdin);
   if (num <= 0)
      printf("Valor inválido!")
}while(num <= 0);
```

Exercícios

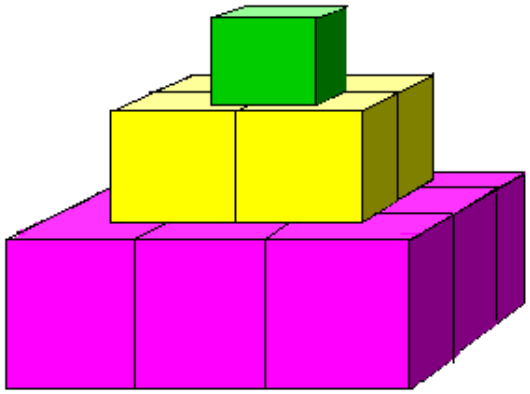
- *I00001200 - Um número inteiro maior do que 1 é primo se ele possui como divisores somente o 1 e ele mesmo. Faça um algoritmo que leia um número (teste a validade da leitura para que seja maior que 1) e verifique se é primo escrevendo: 1 - o número é primo ou 0 - o número não é primo. Dica: Pode-*

se também verificar se um número é primo encontrando seu primeiro divisor maior que 1. Se o primeiro divisor for o próprio número, ele é primo.

- *I00001610 - Faça um algoritmo que leia um valor n (n>=1) correspondente ao número de alunos de uma turma. Após, o algoritmo lê as notas das provas dos n alunos dessa turma. As notas deverão ser lidas até que sejam informadas n notas válidas, ou seja, no intervalo [0, 10], descartando as notas fora desse intervalo. As notas somente poderão ser lidas uma única vez. O algoritmo deve informar qual foi a menor nota e o percentual dos alunos que tiraram a menor nota (que não é, necessariamente, 0). Por exemplo, se o valor lido para n foi 20 e as notas foram 6.0 6.5 8.0 9.0 4.5 3.0 9.0 8.5 4.5 3.0 6.0 3.0 8.0 9.0 4.5 10 9.0 8.5 4.5 3.0 o algoritmo escreve 3.0 e 20, já que quatro (20% de 20) alunos tiraram essa nota.*

- ***Exercício Adicional:** Deseja-se construir uma pirâmide com cubos de 1.5 m3, tal como aparece na figura abaixo. Quantos metros terá de altura a pirâmide, se dispomos de n pedras? Faça um algoritmo que leia o número de pedras N (inteiro e maior que 5) e mostre a quantidade de andares e a altura em metros da pirâmide. Caso sobrem pedras, essa informação deve ser mostrada também. Exemplo: A pirâmide terá X andares, X.X m de altura, e sobraram X pedras.*

***Dica**: Para calcular o número de "andares" (cada andar tem 1.5 metro de altura), deve-se calcular quantas pedras vai em cada andar, um contador para o número de andares e um somador que vai acumulando o total de pedras já gasto até aquele andar. Quando este somador atingir N, deve-se parar. Como provavelmente o somador não vai atingir o valor exato de N, observar onde parar, ou seja, o andar da base tem que ser um andar completo, ainda que sobrem* pedras.



Estrutura de Repetição `for`

A estrutura `for` é muito semelhante às estruturas de repetição vistas anteriormente, entretanto costuma ser utilizada quando se quer um número determinado de ciclos. A contagem dos ciclos é feita por uma variável chamada de **contador**. **Sintaxe:**

```
for(inicialização; condição; incremento)
{
   bloco de instruções
}
```

onde `inicialização` é uma expressão de inicialização do contador, `condição` é uma expressão lógica de controle de repetição, `incremento` é uma expressão de incremento do contador e `bloco de instruções` é um conjunto de instruções a ser executado.

Esta estrutura executa um número determinado de repetições usando um contador de iterações. O contador é inicializado na expressão de `inicialização` **antes** da primeira iteração. Por exemplo: `i = 0;` ou `cont = 20;`. Então, o bloco é executado e **depois** de cada iteração, o contador é incrementado de acordo com a expressão de `incremento`. Por exemplo: `i++` ou `cont -= 2`. Então a expressão de condição é avaliada: se a condição for verdadeira, o `bloco de instruções` é executado novamente e o ciclo recomeça; se a condição for falsa termina-se o laço. Esta condição é, em geral, uma expressão lógica que determina o último valor do contador. Por exemplo: `i <= 100` ou `cont > 0`.

Exemplo: No trecho a seguir, uma variável inicializada com `1` é automaticamente incrementada em `1` e mostrada enquanto for menor ou igual a cinco.

```
for(a = 1 ; a <= 5 ; a++)
   printf("%d", a);
```

Exemplo comparativo: As seguintes instruções são plenamente equivalentes:

```
i = 0;
do{
  bloco de instruções
  i++;
}while(i <= 100);
```

```
for(i = 0; i <= 100; i++)
{
   bloco de instruções
}
```

Exercícios

- *I00001700 - Faça um algoritmo que leia, para 10 pessoas, seu peso e altura e escreva o peso e a altura da pessoa mais alta.*

- *I00002300 (adaptado) - Um número perfeito é o número que é igual à soma de seus divisores, exceto o próprio número (ex: 6 = 1 + 2 + 3; 28=1+2+4+7+14). Faça um algoritmo que gere e mostre os quatro primeiros números perfeitos. Dica: utilizar um while externamente, com uma variável contadora que encerra quando chegar no quarto número perfeito, e um for internamente, para cada valor a ser testado.*

- *I00002350 - Faça um algoritmo que leia, para um funcionário, o valor que ganha por hora e 30 pares de valores (hora de entrada e hora de saída, inteiros, sem minutos) e calcule o quanto ganhou no mês. O funcionário não pode trabalhar mais de 23 horas seguidas e pode iniciar em um dia e terminar no dia seguinte. Para o funcionário deve ser escrito o quanto ganhou no mês.*

- *I00002700 - O fatorial de um número N (representado por N!) é o produto de todos os números de 1 a N. Assim, 4! = 1 x 2 x 3 x 4 = 24. Faça um algoritmo que leia um número N e escreva seu fatorial.*

- *Para a próxima aula, fazer a TDE 6.*

Vetores

Em muitas aplicações, é necessário trabalhar com conjuntos de dados que são **semelhantes em tipo**, por exemplo: o conjunto das alturas dos alunos de uma turma, ou um conjunto de seus nomes. Nestes casos, seria conveniente poder colocar estas informações sob um mesmo conjunto, e poder referenciar cada dado individual deste conjunto por um índice. Em programação, este tipo de estrutura de dados é chamado de **vetor** (ou *array*, em inglês) ou, de maneira mais formal, **estruturas de dados homogêneas**. **Exemplo:** A maneira mais simples de entender um vetor é através da visualização de um **lista** de elementos com um nome coletivo e um índice de referência aos valores da lista.

```
n   nota
0   8.4
1   6.9
2   4.5
3   4.6
```

Nesta lista, `n` representa um número de referência e `nota` é o nome do conjunto. Assim, podemos dizer que a segunda nota é `6.9`, ou representar `nota[1] = 6.9`

Declaração de vetores

Em C, um vetor é um conjunto de variáveis de um **mesmo tipo** que possuem um nome identificador e um índice de referência. **Sintaxe:**

```
tipo nome[tam];
```

onde `tipo` é o **tipo** dos elementos do vetor: `int`, `float`; `nome` é o **nome** identificador do vetor, com as regras de nomenclatura de vetores iguais às das variáveis, e; `tam` é o tamanho do vetor, isto é, o número de elementos que o vetor pode armazenar. **Exemplos:**

```
int idade[100];   // declara um vetor chamado 'idade' do tipo
                  // 'int' que recebe 100 elementos.
float nota[25];   // declara um vetor chamado 'nota' do tipo
                  // 'float' que pode armazenar 25 números.
```

Referência a elementos de vetor

Cada elemento do vetor é referenciado pelo **nome** do vetor seguido de um **índice** inteiro. O **primeiro** elemento do vetor tem índice `0` e o **último** tem índice `tam-1`. O índice de um vetor deve ser **inteiro**.

Exemplo: Algumas referências a vetores:

```
int i = 7;
float valor[10];                /* declaração de vetor */
valor[1] = 6.645;
valor[i] = 7.645;
```

Inicialização de vetores

Assim como é possível inicializar variáveis (por exemplo: `int j = 3;`), pode-se inicializar vetores. **Sintaxe:**

```
tipo nome[tam] = {lista de valores};
```

onde `lista de valores` é uma lista, separada por vírgulas, dos valores de cada elemento do vetor.

Exemplo:

```
int dia[7] = {12,30,14,7,13,15,6};
float nota[5] = {8.4,6.9,4.5,4.6,7.2};
```

Opcionalmente, é possível inicializar os elementos do vetor enumerando-os um a um. **Exemplo:** A duas inicializações a seguir são possíveis:

```
int cor_menu[4] = {1,2,3,4};
```

ou

```
int cor_menu[4];
cor_menu[0] = 1;
cor_menu[1] = 2;
cor_menu[2] = 3;
cor_menu[3] = 4;
```

Exercícios

- *V00000100 - Escrever um algoritmo que lê um vetor v(8). Conte, a seguir, quantos valores de V são negativos e escreva esta informação.*

- *V00000200 - Faça um algoritmo que leia 10 valores quaisquer e armazene em um vetor. Após, os escreva na ordem contrária à que foram digitados.*

- *V00000300 - Faça um algoritmo que leia 9 valores inteiros maiores que zero para um vetor v e escreva primeiramente todos os números pares digitados e em seguida os números ímpares.*

- *V00000400 - Escrever um algoritmo que lê um vetor de inteiros maiores que zero para um vetor v(7). Conte, a seguir, quantos valores de v são primos e escreva esta informação. Além de contar, o algoritmo deve mostrar os primos.*

- *V00000500 - Faça um algoritmo que leia dez números inteiros, armazenando-os em um vetor. Após, escreva a posição de cada número menor que zero desse vetor.*

- *V00000800 - Escrever um algoritmo que lê um vetor v(15) e o escreve. Encontre, a seguir, o maior elemento de C e o escreva.*

- *V00000900 - Escrever um algoritmo que lê um vetor v(10) e o escreve. Encontre, a seguir, o menor elemento e a sua posição no vetor v e escreva: "O menor elemento é ... e a sua posição é ... ".*

- *Escrever um algoritmo que lê 10 valores inteiros em um intervalo de 0 a 100, armazenando em um vetor. Após, percorra o vetor para contar quantos deles estão em cada um dos intervalos [0, 25], (25, 50], (50, 75], (75, 100], mostrando a informação.*

- *A série de Fibonacci tem como dados os 2 primeiros termos da série, que são respectivamente 0 e 1. A partir deles, os demais termos são construídos pela regra:*

  $t_n = t_{n-1} + t_{n-2}$

  Escrever um algoritmo que gera os 10 primeiros termos da série de Fibonacci e armazena em um vetor. Após, calcular a soma destes termos e mostrar juntamente com o vetor.

- *Escrever um algoritmo que lê 10 valores para o vetor v, todos inteiros e positivos. Calcular e mostrar, a seguir, a média aritmética dos valores do vetor, a quantidade de valores pares, a quantidade de valores ímpares, a percentagem de valores pares e a percentagem de valores ímpares.*

- *Escrever um algoritmo que lê 10 valores para o vetor n, todos inteiros e positivos. Após, para cada valor lido, mostrar a tabuada de 1 até n de n.*

  $1 \times n = n$
  $2 \times n = 2n$
  $\vdots \quad \vdots \quad \vdots \quad \vdots$
  $n \times n = n^2$

- *Escrever um algoritmo que lê 10 valores para o vetor v, todos inteiros e positivos. Após, percorrer o vetor e, se o valor for par, verificar quantos divisores possui e mostrar esta informação. Se o valor for ímpar e menor que 12 calcular e mostrar o fatorial do valor. Se o valor for ímpar e maior ou igual a 12 calcular e mostrar a soma dos inteiros de 1 até o valor.*

Métodos de Ordenação

Existem três métodos conhecidos de ordenação itens em uma lista (um vetor, por exemplo):

1. ORDENAÇÃO POR TROCAS (Bubble Sort)
2. ORDENAÇÃO POR SELEÇÃO (Selection Sort)
3. ORDENAÇÃO POR INSERÇÃO (Insertion Sort)

Vamos conhecer a primeira, a ordenação por trocas (bubble sort). A estratégia usada para ordenar os itens de uma lista é comparar pares de itens consecutivos e permutá-los, caso estejam fora de ordem. Se a lista for assim processada, sistematicamente, da esquerda para a direita, o maior valor (ou menor) deslocado para a última posição da lista. Com várias passagens, isso fará com que a lista fique ordenada.

Exemplo: Confira a ordenação completa de uma lista pelo método bolha. A variável i indica a fase da ordenação e j indica a posição do par de itens consecutivos que serão comparados e, eventualmente, permutados.

fase	*i*	*j*	v[0]	v[1]	v[2]	v[3]	v[4]
1ª	1	0	**46**	**39**	55	14	27
		1	39	**46**	**55**	14	27
		2	39	46	**55**	**14**	27
		3	39	46	14	**55**	**27**
			39	46	14	27	**55**
2ª	2	0	**39**	**46**	14	27	**55**
		1	39	**46**	**14**	27	**55**
		2	39	14	**46**	**27**	**55**
			39	14	27	**46**	**55**
3ª	3	0	**39**	**14**	27	**46**	**55**
		1	14	**39**	**27**	**46**	**55**
			14	27	**39**	**46**	**55**
4ª	4	0	**14**	**27**	**39**	**46**	**55**
			14	**27**	**39**	**46**	**55**

Tente, a partir do exemplo, construir um algoritmo para ordenação de vetores.

Exercícios

- *V00001900 - Faça um algoritmo que leia um vetor X(10) e ordene seus elementos em ordem crescente.*

- *V00002000 - Faça um algoritmo que leia, para cada pessoa de um conjunto de 10 pessoas, o seu peso (inteiro) e altura (real), e escreva a lista de pesos e alturas em ordem decrescente de altura. Os dois dados referentes a cada pessoa devem ser lidos juntos (ou seja, o algoritmo não deve ler todos os pesos e em seguida todas as alturas).*

- *Faça um algoritmo que leia 10 valores inteiros para um vetor v de 20 posições que ocuparão as 10 primeiras posições. Ordene, a seguir, os 10*

elementos em ordem crescente. Após, leia 10 valores inteiros, um por vez, e insira-os nas posições adequadas do vetor v, de forma que o mesmo continue ordenado em ordem crescente, mostrando-o ao final.
Observação: Esse é um exemplo de Inserção Ordenada, onde se insere elementos em um vetor ordenado, deixando-o ordenado.

Métodos de Pesquisa (Busca)

Dentre os métodos existentes, iremos ver o método de BUSCA BINÁRIA. Se não sabemos nada a respeito da ordem em que os itens aparecem na lista, o melhor que podemos fazer é uma busca linear. Entretanto, se os itens aparecem ordenados, podemos usar um método de busca muito mais eficiente, conhecido como busca binária.

Esse método é semelhante àquele que usamos quando procuramos uma palavra num dicionário: primeiro abrimos o dicionário numa página aproximadamente no meio; se tivermos sorte de encontrar a palavra nessa página, ótimo; senão, verificamos se a palavra procurada ocorre antes ou depois da página em que abrimos e então continuamos, mais ou menos do mesmo jeito, procurando a palavra na primeira ou na segunda metade do dicionário...

Como a cada comparação realizada o espaço de busca reduz-se aproximadamente à metade, esse método é denominado busca binária.

Suponha que uma lista de valores esteja armazenada num vetor e que o valor procurado x esteja aproximadamente no meio dele. Então temos três possibilidades:

x = meio da lista: nesse caso, o problema está resolvido;
x < meio da lista: então x deverá ser procurado na primeira metade; e
x > meio da lista: então x deverá ser procurado na segunda metade.

E assim sucessivamente para as demais metades encontradas.

Exemplo de algoritmo de Busca binária num vetor de inteiros:

```
int v[30], x;   // x é o valor buscado;

int i, f, m, achou;
i = 0;
f = 29;  //ultimo índice do vetor

achou = 0;   // se não achar, ficará 0
while( i <= f ) {
   m = (i+f)/2;
   if( x == v[m] ) {
       printf("Achou no índice %d",m);
       achou = 1;    // se achar, troca para 1
       break;   // encerra o laço
   }else if( x < v[m])
       f = m-1;
   else
```

```
        i = m+1;
}

if (achou == 0)  // se não achou
    printf("Elemento não encontrado");
```

Exercício

- *Faça um algoritmo que leia um vetor X(10) e ordene seus elementos em ordem crescente. Após, leia um valor e faça a busca dele a partir da busca binária, mostrando em que posição se encontra.*