UNIVERSIDADE FEDERAL DO PARANÁ

ANA PAULA GOHARA

FERNANDO CLAUDECIR ERD

GABRIEL CORREIA SCHONEBORN

GISELE DE PAULA MARTINS

GUILHERME LUIS SALKOVISKI

MATHEUS FELIPE BOSA

PAOLO ANDREAS STALL RECHIA

RENAN PRANDO DAVANZO

C++: UMA ABORDAGEM MINIMALISTA

CURITIBA

2016

ANA PAULA GOHARA

FERNANDO CLAUDECIR ERD

GABRIEL CORREIA SCHONEBORN

GISELE DE PAULA MARTINS

GUILHERME LUIS SALKOVISKI

MATHEUS FELIPE BOSA

PAOLO ANDREAS STALL RECHIA

RENAN PRANDO DAVANZO

C++: UMA ABORDAGEM MINIMALISTA

Objeto de aprendizagem desenvolvido por ensaio colaborativo para o ensino da linguagem de programação C++ aos discentes das disciplinas CI208 / CI180 – Programação de Computadores da Universidade Federal do Paraná.

Orientador: Prof. M. Setembrino Soares Ferreira Junior.

Vice-orientadores: Andrey Ricardo Pimentel; Armando Luiz Nicolini Delgado; Carmem Satie Hara; Guilherme Derenievicz; Leôncio Saraiva Madruga; Luciano Silva; Luis Carlos Erpen de Bona; Marcos Alexandre Castilho; Renato Carmo; Paulo Ricardo de Almeida; Peter Perroni; Ricardo Tavares de Oliveira; Roberto Pereira; Wagner Zola.

CURITIBA

2016

O fim da arte e da educação é substituir a natureza e completar aquilo que ela apenas começou.

Aristóteles

PREFÁCIO

Universidade Federal do Paraná, 08 de dezembro de 2016.

C++: uma abordagem minimalista é resultado do ensaio colaborativo entre monitores e docentes da disciplina CI208 - Programação de Computadores, da Universidade Federal do Paraná, cujo processo de elaboração foi premiado na 15ª edição do ENAF (Encontro das Atividades Formativas), que integrou a 8ª SIEPE (Semana de Ensino Pesquisa e Extensão) da UFPR, realizada em outubro de 2016.

Em harmonia com a característica atual da disciplina, introdutória e de uso prático, o material didático elaborado tem foco na formação objetiva e minimalista necessária para interessados no desenvolvimento de programas na linguagem C++.

Na forma de apostila digital, de acesso gratuito, pretende servir de apoio didático para alunos e docentes, versando sobre aspectos de teoria e prática, com aplicações, na forma de exercícios propostos e resolvidos. Trata-se, pois, de preparar-se para resolver problemas práticos que poderão aparecer ao longo de experiências estudantis e profissionais.

Como resultado de um projeto apresentado num encontro de extensão universitária, abrange os 5 Princípios Extensionistas buscados por atividades desta natureza na Universidade Federal do Paraná, ou seja: Impactos e Transformação, Interação dialógica, Interdisciplinaridade, Indissociabilidade Ensino, Pesquisa e Extensão e Impacto na Formação dos estudantes.

(1) Impactos e transformação e (5) Impacto na formação do publico alvo: pelas características deste objeto de aprendizagem, objetivo e minimalista, há impacto direto na formação de alunos, de maneira objetiva e sem ater-se a aspectos eventualmente questionáveis;

(2) Interação dialógica: *C++: uma abordagem minimalista* é um produto destinado a sofrer evoluções, pela participação do público-alvo, em um processo de melhoria contínua. Por ser armazenado em arquivo digital, há facilidade de acesso para alterações, adequações, correções, acréscimos, ou seja, para o aceite de contribuições e sugestões de ajustes por docentes e discentes, as quais podem ser implementadas de maneira simples e com ações consequentes mais ágeis;

(3) Interdisciplinaridade: *C++: uma abordagem minimalista* é fruto de trabalho caracterizado por sua interdisciplinaridade, uma vez que a disciplina CI208 (PROGRAMAÇÃO DE COMPUTADORES) pertence ao ciclo básico de cursos de engenharia e de ciências exatas (Engenharia Cartográfica, Engenharia Civil, Engenharia Elétrica, Engenharia Química, Física, etc.), e foi desenvolvido de forma integrada por 8 alunos de diversos cursos de graduação, em exercício de monitoria;

(4) Indissociabilidade entre ensino, pesquisa e extensão: *C++: uma abordagem minimalista* é um produto destinado ao ensino, fruto de pesquisa e voltado também para a extensão, fortalecendo este princípio.

(5) Impacto na formação: *C++: uma abordagem minimalista* representou grande oportunidade de prática introdutória à docência, como exercício didático de monitores do Programa de Iniciação à Docência de 2016 (PID 2016) via escrita, oral e no registro e treinamento para a disseminação de conhecimento. O produto pretende oportunizar o

fortalecimento de aspectos teóricos, metodológicos e de cidadania para o público-alvo interessado no tema.

Por fim, e por todas as características discorridas, espera-se que o uso deste material como ferramenta para a disseminação, padronização e uniformização do lecionar e assistir a interessados na linguagem de programação C++ seja atingido.

Os autores

LISTA DE FIGURAS

FIGURA 1: ARQUITETURA DE VON NEUMANN.....7
FIGURA 2: BUILD.....11
FIGURA 3: RUN.....12

SUMÁRIO

LISTA DE FIGURAS.....5
1CONCEITOS INICIAIS.....8
1.1COMPUTADOR.....8
1.2MÁQUINA DE VON NEUMANN.....8
1.3SISTEMAS DE NUMERAÇÃO.....9
1.4MEDIDAS DE CAPACIDADE DE DISPOSITIVOS DE ARMAZENAGEM DE DADOS 10
1.5COMO É GERADO UM PROGRAMA?.....11
2PROGRAMAS EM C++.....14
2.1HISTÓRIA DO C++.....14
2.2PROGRAMA EM C++.....14
2.3SENTENÇAS.....15
2.4HELLO WORLD.....16
3VARIÁVEIS E CONSTANTES.....18
3.1NOMES DE VARIÁVEIS.....18
3.2TIPOS DE DADOS.....19
3.3EXEMPLOS DE ESPECIFICAÇÃO.....19
3.3.1CONSTANTES.....19
3.3.2VARIÁVEIS.....19
4EXPRESSÕES.....21
4.1EXPRESSÕES ARITMÉTICAS.....21
4.2EXPRESSÕES RELACIONAIS (COMPARATIVAS).....22
4.3EXPRESSÕES LÓGICAS.....23
5MANIPULAÇÃO DE VARIÁVEIS.....25
5.1ATRIBUIÇÃO.....25
5.2ENTRADA E SAÍDA DE DADOS.....26
5.2.1ENTRADA.....26
5.2.2SAÍDA / MOSTRA EM TELA.....27
6ESTRUTURA DE CONTROLE DE SELEÇÃO (*IF*() … *ELSE*).....30
7ESTRUTURAS DE CONTROLE DE REPETIÇÃO.....33
7.1*WHILE*().....33
8FUNÇÕES.....38

8.1ESTRUTURA....39
8.2SAÍDA DE DADOS DE FUNÇÕES....42
8.3CORPO DAS FUNÇÕES....42
8.4CHAMADA (USO) DE FUNÇÕES....43
8.5PROTÓTIPOS....45
8.6ESPECIFICAÇÃO DE FUNÇÕES....46
8.7EXEMPLOS....47
9VETORES E MATRIZES....53
9.1VETORES....53
9.1.1FUNÇÕES ENVOLVENDO VETORES....56
9.1.2PROBLEMAS TÍPICOS ENVOLVENDO VETORES....58
9.2MATRIZES....59
9.2.1EXTENSÃO DE CONCEITOS DE VETORES A MATRIZES....61
9.2.2PROBLEMAS TÍPICOS ENVOLVENDO MATRIZES....62
APÊNDICE 1 – TABELA ASCII....64
APÊNDICE 2 – MÁQUINA VIRTUAL....65
APÊNDICE 3 – TIPOS DE DADOS....75
APÊNDICE 4 – OPERADORES DE ATRIBUIÇÃO ALTERNATIVOS....76
APÊNDICE 5 – PRECEDÊNCIA DE OPERADORES....77
APÊNDICE 6 – ESTRUTURA DE REPETIÇÃO *FOR*....78
EXERCÍCIOS: 1 – EXPRESSÕES....79
EXERCÍCIOS: 2 – SELEÇÃO....93
EXERCÍCIOS: 3 – REPETIÇÃO....111
EXERCÍCIOS: 4 – FUNÇÕES COM RETORNO POR VALOR....121
EXERCÍCIOS: 5 – FUNÇÕES COM PARÂMETROS DE SAÍDA....130
EXERCÍCIOS: 6 – VETORES....135
EXERCÍCIOS: 7 – MATRIZES....157
EXERCÍCIOS: 8 – QUESTÕES DE PROVAS....168

1 CONCEITOS INICIAIS

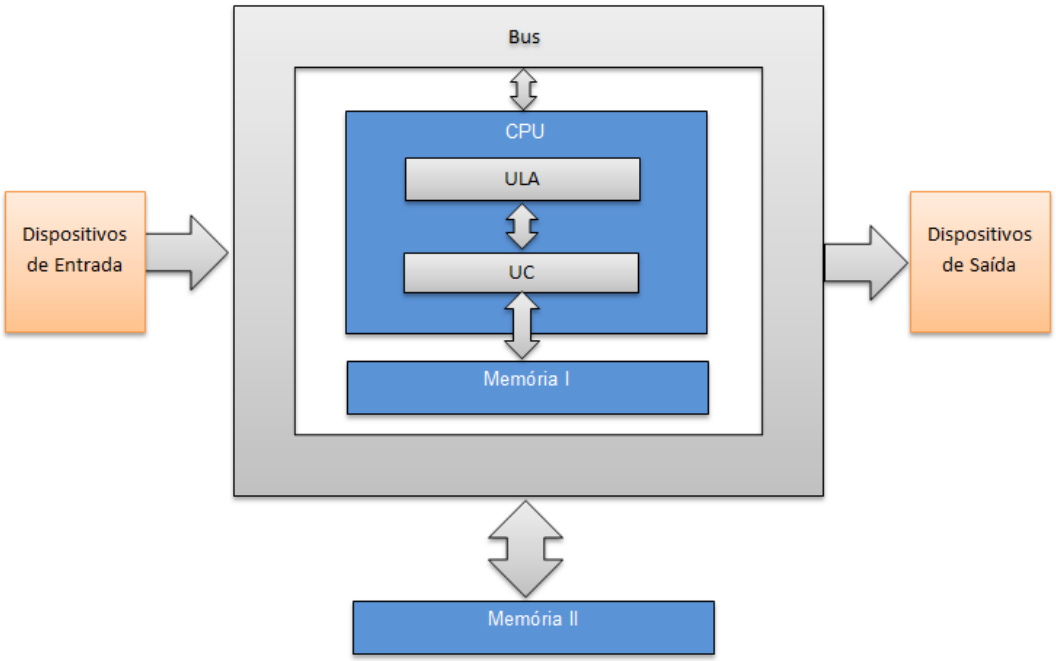
1.1 COMPUTADOR

Um computador, independente de sua geração de desenvolvimento, é um equipamento utilizado para processar dados, capaz de realizar quatro funções básicas: entrada, processamento, armazenagem e saída de dados.

1.2 MÁQUINA DE VON NEUMANN

A máquina de Von Neumann é uma proposta de arquitetura de computadores construída de modo a conter juntos dados e códigos, ou seja, memórias de dados e de programas.

FIGURA 1: ARQUITETURA DE VON NEUMANN



Fonte: Os autores

A máquina proposta por Von Neumann reúne os seguintes componentes:

- **Bus:** Conjunto de ligações físicas que permitem a interligação entre os elementos do sistema, tais como cabos e pistas de circuitos;
- **Unidade Central de Processamento (CPU)**

  - Unidade de Controle (UC): determina a sequência das instruções a serem executadas. Também gera os sinais de controle para as outras unidades;

  - Unidade Aritmética e Lógica (ULA): execução de cálculos (operações aritméticas e lógicas);

- **Memória I (Primária ou Principal)**: Armazenamento interno de dados, composto por circuitos em placas, fixados à placa-mãe, pode conter dados e instruções, estas interpretadas como partes de programas a serem executadas pela CPU;
- **Memória II (Secundária ou Auxiliar)**: Armazenamento externo de dados. Exemplos: Disco rígido, dispositivos USB, etc;
- **Dispositivos de entrada***:* Responsáveis por coletar e codificar os dados inseridos pelo usuário. Exemplos: teclado, mouse, pen drive, microfone, etc;
- **Dispositivos de saída:** Decodificam os dados para que possam ser entendidos pelo usuário. Exemplos: monitor de vídeo, caixa de som, impressora, pen drive, etc.

1.3 SISTEMAS DE NUMERAÇÃO

Assim como no sistema decimal, de base 10, utilizado em nosso dia-a-dia, os dígitos vão de 0 a 9, no sistema de base 2 (binário) temos um sistema de numeração formado por apenas dois algarismos: 0 (zero) e 1 (um). Ou seja, admite duas possibilidades sempre opostas, como: nada / tudo, desligado / ligado, ausência / presença, etc. Sua origem coincidiu com o uso, nos primeiros computadores, de anéis de ferrite (material magnetizável) utilizados na construção de memórias de núcleos. O conceito teve origem no magnetismo, onde se sabe que um condutor sujeito a uma corrente contínua induz um campo eletromagnético ao seu redor; assim, os anéis de ferrite magnetizados em um sentido ou outro eram entendidos como zeros ou uns.

A conversão de números representados na base 10 para a base 2 faz-se por divisões sucessivas do número decimal por 2. Em cada divisão os restos,

necessariamente, serão zeros ou uns. Lidos de trás para frente o último quociente das divisões e os sucessivos restos obtém-se a representação binária de um número na base 10.

Exemplo:

Número na base 10: 8
8/2 = 4, resto 0;
4/2 = 2, resto 0;
2/2 = 1, resto 0.
Assim, $(8)_{10} = (1000)_2$.

A conversão de números representados na base 2 para a base 10 faz-se somando os valores intrínsecos referentes a cada posição dos dígitos do número binário. Seja um número binário qualquer, representado por $d_n d_{n-1} d_{n-2} ... d_i ... d_2 d_1 d_0$, com $d_i = 0$ ou 1 para todo i = n, n-1, n-2, …, 2, 1, 0; seu valor na base 10 é obtido pela soma dada pelo valor do dígito naquela posição multiplicado por seu valor instrínseco, ou seja, 2^n, 2^{n-1}, 2^{n-2}, …, 2^2, 2^1, 2^0.

Exemplo:

Número na base 2: 1001
$1x2^3 + 0x2^2 + 0x2^1 + 1x2^0 = 9$
Assim, $(1001)_2 = (9)_{10}$.

1.4 MEDIDAS DE CAPACIDADE DE DISPOSITIVOS DE ARMAZENAGEM DE DADOS

- ***Bit (Binary Digit)****:* Menor unidade de informação que pode ser armazenada ou transmitida na comunicação de dados. O *bit* pode assumir somente dois valores, ou 0 ou 1, coincidentes com os algarismos do sistema binário;
- ***Byte****:* Um *byte* é uma estrutura utilizada em computação composta por um conjunto de *bits*. Vários tamanhos de agrupamentos apareceram ao longo do tempo, de 6 em 6, 8 em 8, 11 em 11, etc. O conjunto que se consolidou e que hoje se entende por *byte* foi o do agrupamento de 8 em 8 *bits.* O preenchimento de um *byte* com *bits* permite 256 combinações diferentes, de “00000000” até “11111111”, passando por “00000001”, “00000010”, “00000011”, etc. A cada combinação obtida atribuiu-se um caractere determinado, compondo uma tabela denominada ASCII (*American Standard Code for Information Interchange*). Com isto, um *byte* do mundo da informática corresponde a um caractere do mundo real. Utiliza-se o *byte* e seus múltiplos (K – Kilo = 1.024; M – Mega = 1.000.000; G – Giga = 1.000.000.000; T – Tera = 1.000.000.000.000) para especificar o tamanho de memórias ou a

capacidade de armazenamento de um dispositivo, independentemente do tipo de dado armazenado.

1.5 COMO É GERADO UM PROGRAMA?

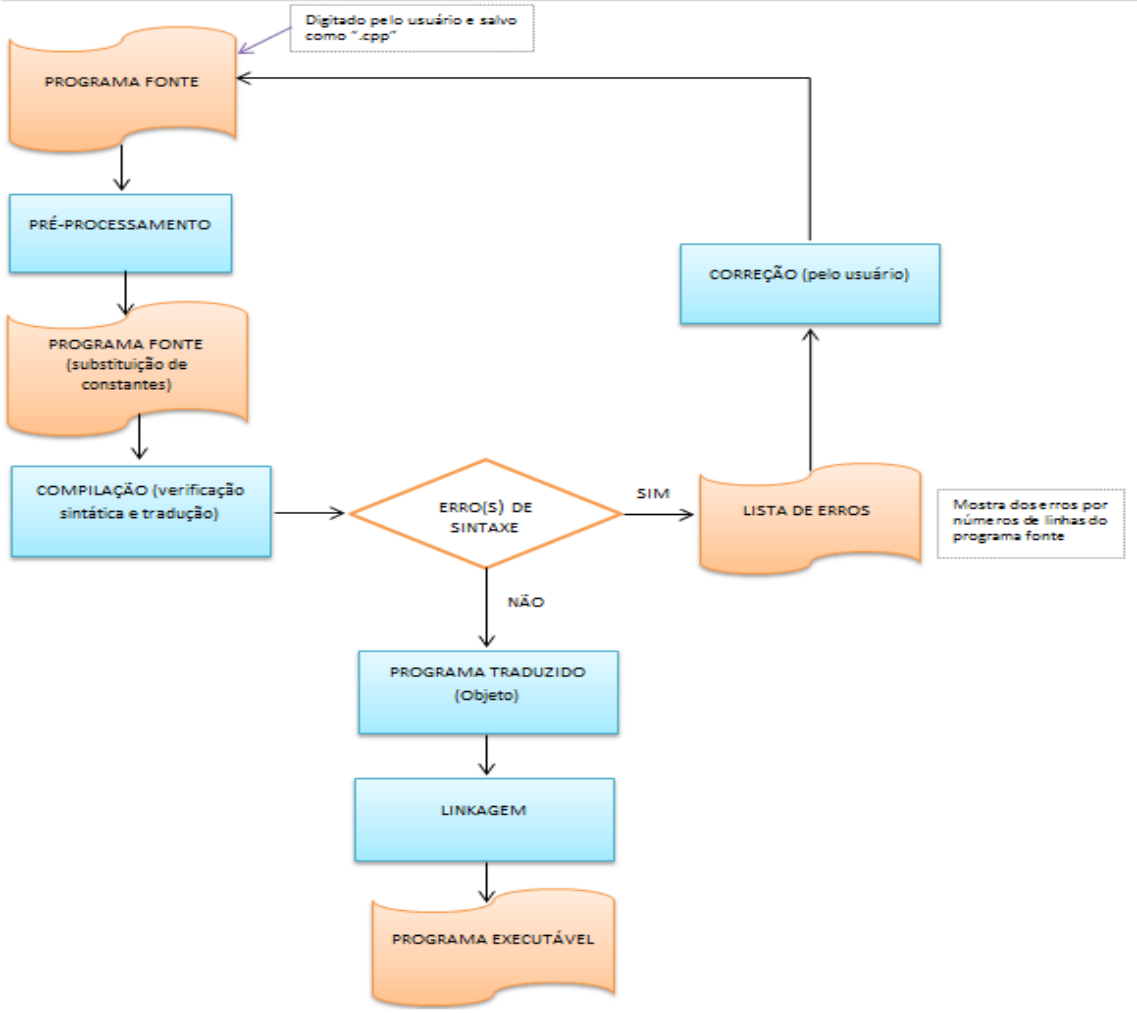
O código de um programa que escrevemos, chamado programa fonte, por si só são apenas linhas de caracteres, mas quando o computador entende essas linhas e gera um programa executável, esse processo é chamado de compilação.

Após escrevermos um código somos obrigados a compilá-lo para obter um programa executável. Visualmente para o usuário pode parecer simples (muitas vezes só clicar em um determinado ícone ou digitar uma linha de comando), porém são 4 as etapas que ocorrem na geração do arquivo executável, derivado do programa fonte digitado em um editor de texto. Essas etapas são: o pré-processamento, verificação sintática, tradução e *linkagem*.

- **Pré-processamento***:* Nesta etapa ocorrem as substituições de constantes, como por exemplo: foi definida uma constante chamada TAM que tem o valor 10. Nesse processo o compilador vai substituir todas as ocorrências de TAM por 10. Também nesta etapa ocorre a disponibilização de módulos com a inclusão solicitada pelo usuário via declarações iniciadas com “#include”;
- **Verificação Sintática***:* A parte mais odiada por muitos, nesse processo ocorre a verificação sintática do programa, onde é checado se falta um ponto e vírgula, chaves, parênteses, um comando escrito errado, etc. Assim que são identificados, esses erros são mostrados na tela;
- **Tradução***:* Aqui ocorre a transformação do código fonte na linguagem que o processador entende (essa linguagem varia de processador para processador), onde uma linha em C pode virar em torno de 10 em código de processador. Nesse ponto estamos em uma linguagem que o computador consegue decifrar o código, porém ainda não é possível executá-lo;
- ***Linkagem****:* Onde ocorre a ligação do código traduzido com bibliotecas chamadas com as declarações “*#include”*, como *iostream*, *cmath* e outras.

Passadas estas 4 etapas, o código gerado é um programa executável, conforme a figura a seguir.

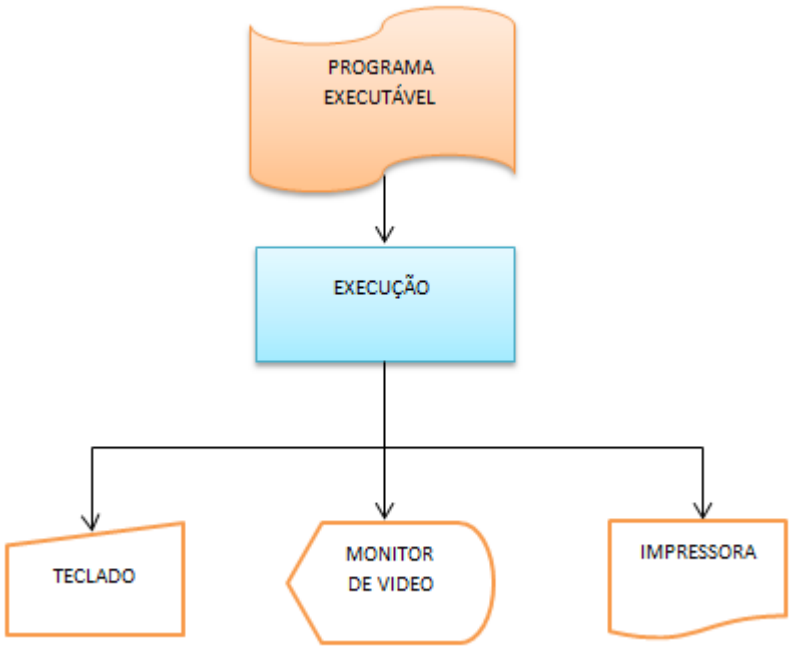
FIGURA 2: *BUILD*



Fonte: Os autores

O programa executável obtido pode então ser submetido à execução propriamente dita, conforme a figura abaixo.

FIGURA 3: *RUN*



Fonte: Os autores

Referências

Hardware – Dispositivos de Entrada e/ou Saída. Disponível em: <http://www.ufpa.br/dicas/mic/mic-e-s.htm>. Acesso em 15 de setembro de 2016.

2 PROGRAMAS EM C++

2.1 HISTÓRIA DO C++

A linguagem de programação C++ teve seu início como uma versão expandida da linguagem C, sendo Bjarne Stroustrup o inventor das primeiras extensões C++, no ano de 1979, em New Jersey.

A maioria dessas adições feitas ao C por Stroustrup tiveram por finalidade incorporar conceitos da programação orientada a objetos à linguagem. Por isso, inicialmente chamou-se a nova linguagem de "C com classes". No entanto, em 1983 o nome passou a ser C++.

Embora C fosse uma das linguagens preferidas e mais usadas para programação profissional em todo o mundo, o desenvolvimento de C++ fez-se necessário devido ao aumento da complexidade na programação empregada na época. Os programas de computador mostravam-se cada vez maiores e mais elaborados.

Apesar de ser uma excelente linguagem de programação, C tem seus limites. Uma vez que um programa atinja valores de 25.000 a 100.000 linhas de código, este se torna tão complexo que é difícil analisá-lo como um todo; isto é, a manutenção se torna um pesadelo, e é neste ponto que aparece o C++. A essência do C++ é permitir que programadores compreendam e gerenciem programas cada vez mais complexos.

2.2 PROGRAMA EM C++

Um programa em C++ é um conjunto de uma ou mais funções.

Sempre obrigatória, a função principal em um programa é a função *main().* A execução de qualquer programa sempre tem sua interpretação iniciada pelos códigos nela descritos.

Estrutura geral de um programa em C++

```
#include <iostream>
        using namespace std;
Declaração de constantes
Funções
main()
{
Declaração de variáveis
Sentenças
...
}
```

2.3 SENTENÇAS

Este termo é usado para referir-se aos comandos propriamente ditos da linguagem. Quando uma ação em um programa puder ser configurada com um único comando, estamos diante de uma sentença simples.

Exemplo:

```
x = x + 3;          // incrementa a variável x com 3 unidades e atribui o resultado
                    // a ela mesma
```

Caso uma ação necessite de vários comandos agrupados, haveremos de escrever uma sentença composta, ou seja, escrever os comandos dentro de um conjunto de chaves ({…}).

Exemplo:

```
{
        x = x + 3;
        y = y + 1;
        cout << x + y << endl;
}
```

Observe que a forma como se escreve a função principal de um programa (main()) é um exemplo de utilização de sentença composta.

2.4 HELLO WORLD

O programa de computador mais simples a ser criado quando se estuda uma linguagem de programação é chamado de "*Hello World!*" ou "Olá, Mundo!". Eis abaixo o correspondente na linguagem C++:

```
#include <iostream>
        using namespace std;
main()
{
        cout << "Olá, mundo!" << endl;
}
```

Esse programa é responsável por simplesmente mostrar na tela do computador a mensagem "Olá, mundo!".

Vale, neste momento, destacar algumas palavras deste código que se mostrarão presentes em programas futuros:

- *#include <iostream>*

Parte do cabeçalho do programa onde "*include*" (incluir) diz para o preprocessador buscar algumas instruções (bibliotecas) para utilizar durante a execução. No caso apresentado, foi declarada a biblioteca "*iostream*", vinda de *in+out+stream* (fluxo de entrada e saída). Ela define como entrada padrão (por onde o usuário coloca dados) o teclado e, como saída padrão (por onde o usuário visualiza os dados processados), o monitor de vídeo;

- *using namespace*

Esse comando define um *namespace* ou "espaço de nomes". Um *namespace* permite a definição de estruturas, funções, etc, que estarão vinculadas a ele. Por definição, a linguagem C++ utiliza o *namespace std* para definir todas as funções da biblioteca padrão.

- *main()*

É a função principal de entrada do programa a partir do sistema operacional. Por isso ela é obrigatória em qualquer programa. Todo programa tem sua execução iniciada nesta função. Caso não existisse, não haveria entrada para que o sistema iniciasse a execução do programa. O uso do Code::Blocks necessita apenas da declaração desta função na forma "*main()*", porém, em algumas implementações de compiladores outras formas alternativas são adotadas: "*int main()", "int main(void)" e "main(void)".* A palavra *void* deriva de "*value of Id";*

- *cout*

O comando direciona uma informação do programa para a saída padrão que, no nosso caso, é o monitor de vídeo. A palavra "*cout*" vem de "*console out*" (saída de console) e seguida de "<<". No exemplo, imprime a mensagem escrita entre aspas;

- *endl*

É uma ferramenta de formatação do que se imprime na tela. Seu significado vem de "*end line*" (fim de linha) e, quando inserida, realiza uma quebra de linha (mudança para a primeira posição da próxima linha);

IMPORTANTE: Em alguns computadores, a janela do programa fecha automaticamente ao final de sua execução, e normalmente queremos continuar vendo a solução gerada pelo programa, mesmo que ele tenha terminado. Uma solução simples é usar o comando *system("pause")* no final do programa e, para isso, deve ser incluída a biblioteca *cstdlib* no inicio do programa. Eis um exemplo de uso desse comando:

```
#include <iostream>
#include <cstdlib>          // inclui biblioteca para uso do comando system("pause")
        using namespace std;
main()
{
        cout<<"Hello world!"<<endl;
        system("pause");    // comando para pausar o programa
}
```

Referência

Introdução – C++. Disponível em: <http://www.tiexpert.net/programacao/c/introducao-cpp.php>. Acesso em 30 de agosto de 2016.

3 VARIÁVEIS E CONSTANTES

Variável é um elemento da programação utilizado para guardar valores que o programa pode utilizar e manipular ao longo da execução. Ao declarar-se uma variável, o computador reserva um espaço na memória para um tipo determinado de dado (podendo ser um número inteiro, real, um caractere, etc).

Para o caso de constantes, não há armazenamento em memória envolvido. Cria-se sinônimos de valores (inteiros, reais, ...) através de nomes definidos para as constantes. Como o nome sugere, esses valores não podem ser modificados durante a execução do programa. As constantes são interpretadas numa etapa de pré-processamento na rotina de compilação dos programas. Os valores declarados para as constantes (via #define) são substituídos por seus nomes onde ocorrerem em um programa.

3.1 NOMES DE VARIÁVEIS

Nomes são dados às variáveis para referenciar os valores utilizados ao longo do programa. Existem algumas regras para a declaração dos nomes de variáveis:

- os nomes só podem conter letras, dígitos e "_" (*underline):* não utilizar quaisquer outros símbolos;
- o "_" (*underline*) é interpretado como uma letra;
- letras maiúsculas são interpretadas pelo programa como diferentes das letras minúsculas (ex: S ≠ s);
- o primeiro caractere deve ser sempre uma letra ou "_";
- não podem ser usadas palavras reservadas da linguagem de programação (ex: *do, break, return*, etc.) como nomes de variáveis.

Vale ressaltar que é interessante dar nomes às variáveis de modo que possuam alguma relação com o que elas representam, como por exemplo: soma, contador, etc.

3.2 TIPOS DE DADOS

Os dados processados por um programa em C++, no escopo deste trabalho, podem assumir três tipos básicos, que são:

- **int**: Valor inteiro, é o tipo padrão. O tamanho do conjunto que pode ser representado normalmente depende da máquina em que o programa está rodando;
- **float**: Valor em ponto flutuante, de precisão simples. São conhecidos normalmente como números reais;
- **char**: O valor armazenado é um caractere. Caracteres geralmente são armazenados em códigos (ASCII).

3.3 EXEMPLOS DE ESPECIFICAÇÃO

3.3.1 CONSTANTES

Em C++, as constantes são definidas através de comandos chamados de “diretivas define”, logo no início do programa, em seguida à cláusula “using namespace std;” (ver estrutura geral de um programa C++ - seção 2.2).

#define Pi 3.14159, por exemplo, cria a constante de tipo real de nome “Pi” com o valor de 3,14159;

#define Salario_Minimo 880.00 cria a constante real “Salario_Minimo” com valor de 880,00;

#define Tamanho_Semana 7 cria a constante inteira “Tamanho_Semana” com valor de 7;

#define Sexo ‘M’ cria a constante caractere “Sexo” com valor ‘M’.

Observe-se que o tipo de dado das constantes é assumido intrinsicamente, ou seja, a partir do valor a elas declarado. Nos exemplos acima, os tipos float, float, int e char foram assumidos para constantes, respectivamente, real, real, inteira e caractere.

3.3.2 VARIÁVEIS

A especificação de variáveis segue o formato geral:

Declaração de variáveis – formato geral

tipo nome-de-variável-1[, nome-de-variável-2, …, nome-de-variável-n];

Onde:

tipo: é a declaração de um dos tipos básicos – *int*, *float* ou *char*, respectivamente identificando variáveis inteiras, reais ou tipo caractere;

nome-de-variável-1[, nome-de-variável-2, …, nome-de-variável-n]: é uma lista de variáveis, identificadas por seus respectivos nomes, criada de acordo com a necessidade do usuário.

As declarações:

int i, contador;

float area, perimetro, resultado;

char sexo, id_motivo;

ilustram a definição de variáveis inteiras (“i” e “contador”), reais (“area”, “perimetro” e “resultado”) e tipo caractere (“sexo” e “id_motivo”).

Enquanto as constantes têm sua definição em um único local na estrutura geral de um programa, as definições de variáveis podem ocorrer em diversos escopos. Neste trabalho, estas ocorrerão no início da função principal (main()) ou no início de quaisquer funções definidas pelo usuário.

A declaração de uma variável cria uma referência a um local (endereço) de memória. É uma correspondência biunívoca, ou seja, a uma variável corresponde um único endereço de memória e vice-versa. De acordo com o tipo da variável ocorre a ocupação da memória; as de tipo inteiro (*int*) alocam 2 bytes da memória, as reais (*float*) 4 e as do tipo caractere (*char*) apenas 1.

4 EXPRESSÕES

Assim como na matemática, as expressões em C++ são conjuntos do tipo: {operando ; operador ; operando} envolvendo operandos constantes e / ou variáveis e operadores. As expressões são divididas em: aritméticas, relacionais e lógicas, de acordo com o operador utilizado.

4.1 EXPRESSÕES ARITMÉTICAS

São expressões que dão como resultado um determinado valor aritmético, resultado das operações efetuadas.

Operadores Aritméticos:

OPERADOR	SINTAXE	EXEMPLO	RETORNO
Adição	A + B	10+5	15
Subtração	A – B	10-5	5
Multiplicação	A * B	10*5	50
Divisão (inteira)	A / B	10/4	2
Divisão (real)	A / B	10./4	2.5
Resto da Divisão	A % B	10%3	1

Obs: Para se utilizar o operador “%”, resto da divisão inteira, os operandos envolvidos têm de ser do tipo INTEIRO, caso contrário ocorrerá erro de compilação.

Exemplo:

```
#include <iostream>
        using namespace std;
main ( )
{
        int A, B;
        A=8;
        B=5;
        cout << A + B << endl;
        cout << A – B << endl;
        cout << A * B << endl;
        cout << A / B << endl;
        cout << A*1. / B << endl;
        cout << A % B << endl;
```

```
}
```

O programa mostra como resultado:

```
13
3
40
1
1.6
3
```

Pelos resultados gerados, deve-se ter cuidado com o uso de variáveis para resultado de operações. O caso “A / B” é típico: como A e B são inteiros, o primeiro resultado obtido foi 1 pois, como ambos os operandos são inteiros, a aritmética inteira é adotada – nesta aritmética não há frações. No caso “A * 1. / B”, ao multiplicar-se a variável A (inteira) por “1.”, o resultado (parcial) é um número real; dividido por um inteiro (no caso, B), misturaram-se as aritméticas real e inteira – neste caso, a aritmética mais complexa (real) é adotada na interpretação da expressão. Caso desejássemos colocar os resultados obtidos em variáveis, a expressão “A / B” deveria ter seu resultado atribuído a uma variável inteira (resultado = 1). E, para o caso da expressão “A * 1. / B”, a uma variável real (resultado = 1.6).

4.2 EXPRESSÕES RELACIONAIS (COMPARATIVAS)

Retornam para o programa os valores lógicos verdadeiro (1) ou falso (0).

Operadores relacionais:

OPERADOR	SINTAXE	EXEMPLO	RETORNO
Menor que	A < B	10 < 5	Falso
		5 < 10	Verdadeiro
Menor ou igual a	A <= B	10 <= 5	Falso
		10 <= 10	Verdadeiro
Maior que	A > B	10 > 5	Verdadeiro
		5 > 10	Falso

Maior ou igual a	A >= B	10 >= 5	Verdadeiro
		5 >= 10	Falso
Igual	A == B	10 == 5	Falso
		10 == 10	Verdadeiro
Diferente	A != B	10 != 5	Verdadeiro
		10 != 10	Falso

4.3 EXPRESSÕES LÓGICAS

São expressões que combinam resultados de expressões aritméticas e/ou lógicas através de operadores lógicos. São estes:

OPERADOR	REPRESENTAÇÃO
E	&& ou and
OU	| | ou or

Associações de resultados de expressões:

A	B	A or B	A and B
0 (falso)	0 (falso)	0 (falso)	0 (falso)
0 (falso)	1 (verdadeiro)	1 (verdadeiro)	0 (falso)
1 (verdadeiro)	0 (falso)	1 (verdadeiro)	0 (falso)
1 (verdadeiro)	1 (verdadeiro)	1 (verdadeiro)	1 (verdadeiro)

Negação lógica:

Único operador unário utilizado; todos os demais (aritméticos, relacionais ou lógicos) apresentados são binários, ou seja, requerem dois operandos, um à sua esquerda e um à direita para a operação ser efetuada.

OPERADOR	REPRESENTAÇÃO
NÃO	!

!0 -> 1

!1 -> 0

Para a = 1 (verdadeiro) e b = 0 (falso):

!(a and b) -> 1

!(a or b) -> 0

Referências:

DELGADO, A. L. N; **Linguagem C++ - Notas de Aula.** Curitiba, junho de 2012.

Operadores em C e C++. Disponível em: <https://pt.wikipedia.org/wiki/Operadores_em_C_e_C%2B%2B>. Acesso em 30 de agosto de 2016.

5 MANIPULAÇÃO DE VARIÁVEIS

5.1 ATRIBUIÇÃO

Após a declaração de uma variável, seu valor é indefinido (uma sujeira qualquer pré-existente no endereço associado à variável, “only God knows” ou “só Deus sabe”), porque nenhum valor lhe foi atribuído. É, portanto, recomendável atribuir valores iniciais para as variáveis do seu programa e, para fazer isso, duas opções podem ser adotadas: solicitação ao usuário para fazer entrada de dados (via comando “*cin*”); ou atribuindo um valor diretamente à variável através de atribuição (“=”), com o seguinte formato:

Atribuição – formato geral

variável = expressão;

Dessa forma podemos atribuir valores inteiros, reais ou caracteres[1] à variável desejada, conforme seu tipo.

([1]): a atribuição de caracteres é feita com valores entre aspas simples, por exemplo 'A'.

Exemplo:

```
#include <iostream>
        using namespace std;
#define h 10                    //define a constante h de valor 10
#define PI 3.1415               //define a constante PI de valor 3.1415

main()
{
        int x;                  //declara a variável inteira x
        x=3;                    //atribui o valor 3 a x
        float y;                //declara a variável real y
        y=55.14;                //atribui o valor 55.14 a y
        char z;                 //declara a variável tipo caracter z
        z='j';                  //atribui o valor “j” a z
```

```
        int m;          //declara a variável inteira m
        m=h;            //atribui o valor da constante h a m
        float b;        //declara a variável real b
        b=PI;           //atribui o valor da constante PI a b
}
```

5.2 ENTRADA E SAÍDA DE DADOS

Em C/C++ é possível ler uma entrada do usuário através do teclado. Esta operação requer, obrigatoriamente, o envolvimento de uma ou mais variáveis, à(s) qual(is) valor(es) digitados pelo usuário serão atribuídos.

Para que o teclado possa ser utilizado em um programa, a cláusula “#include <iostream>” deve ser codificada no início do mesmo.

Esta declaração avisa ao compilador que a entrada e saída padrão serão utilizadas, ou seja, teclado e monitor de vídeo[2].

([2]): Outras maneiras de uso do monitor de vídeo são apresentadas em apêndice deste capítulo

5.2.1 ENTRADA

Para entrada, temos o comando *cin*, e sua forma geral é:

Comando *cin* – formato geral

cin >> variável-1 [>> variável-2 >> ... >> variável-n];

Funcionamento do comando:

Ao ser executado, o comando *cin* gera uma interrupção no processamento, ficando o cursor parado piscando no monitor de vídeo, aguardando digitação do usuário. Se apenas uma variável for descrita no comando cin, apenas um valor deverá ser digitado (compatível com o tipo da variável) e, em seguida, deverá ser teclado <*enter*>; se mais de

uma variável for descrita no comando cin, tantos valores quantos forem as variáveis deverão ser informados, separados por espaços em branco (“ “) ou por <*enter*>.

Interceptada a digitação do último <*enter*> pelo usuário (ou do único <*enter*>, quando se tratar de uma só variável na lista de variáveis do comando *cin*), o(s) valor(es) digitado(s) é(são) movido(s) para a(s) variável(is) envolvida(s).

Exemplo:

```
#include <iostream>                  // inclui a biblioteca iostream
        using namespace std;

main()
{
        int i;                       // declara a variável inteira i
        cin >> i;                    // deixa disponível ao usuário a possibilidade de
                                     // informar, via teclado, um valor inteiro a atribuir
                                     // à variável i
}
```

5.2.2 SAÍDA / MOSTRA EM TELA

Para saída, temos o comando *cout*, e sua forma geral é:

Comando *cout* – formato geral

cout << conteúdo-1 [<< conteúdo-2 << … << conteúdo-i << ... << conteúdo-n];

onde:

conteúdo-i (i = 1, 2, ..., n) = constante;
ou variável;
ou expressão (aritmética, relacional ou lógica);

ou “o texto que você quiser”;

ou endl (para salto à próxima linha do monitor).

Funcionamento do comando:

Cada conteúdo especificado no comando *cout* tem seu valor exibido no monitor de vídeo.

Exemplo:

```
#include <iostream>                  //declara a biblioteca iostream
        using namespace std;

main()
{
        int x;                                  //declara a variável inteira x
        x = 100;                                //atribui o valor 100 a x
        cout << ”O valor de x eh “;             //texto exibido na tela
        cout << x << endl;                      //valor da variável x é exibido na tela e, a
                                                //seguir, o cursor pula para a próxima linha
                                                //no monitor de vídeo
}
```

Observação:

Recomenda-se o uso da palavra reservada “endl” como último conteúdo do último comando “cout” utilizado pelo programa para que a última linha de mensagens enviadas pelo usuário ao monitor de vídeo não fique emendada com textos padrão que normalmente aparecem ao final da execução de um programa.

Exemplo:

```
#include <iostream>                  //declara a biblioteca iostream
        using namespace std;

main()
{
```

```
        int x;                          //declara a variável inteira x
        cout << ”Ola” << endl;           //imprime na tela o texto e, em
                                         //seguida, pula uma linha
        cout << ”Digite o valor de x: “; //imprime na tela o texto
        cin >> x;                        //usuário entra com um valor para a
                                         //variável x
        cout << ”O valor de x eh “       //imprime o texto, o valor de x e,
                << x << endl;            //em seguida, pula uma linha
}
```

6 ESTRUTURA DE CONTROLE DE SELEÇÃO (*IF*() … *ELSE*)

Na grande maioria dos programas, precisamos selecionar trechos a serem executados ou não. Para isso, é preciso fazer algum tipo de seleção, como comparar valores de variáveis, etc. É para isso que existe o comando ***if*()**.

Comando *if*() – formato geral

if(expressão)
    sentença-1
[*else*
    sentença-2]

Funcionamento do comando:

Quando o comando *if*() é interceptado, a expressão (aritmética, relacional ou lógica) descrita à sua direita é avaliada, ou seja, seu valor é calculado e comparado com o valor 0 (zero).

Caso o valor calculado seja diferente de 0 (zero), o controle do programa passa à execução da sentença-1; caso contrário, e a cláusula "*else*" (por isso os [ ]) estiver descrita, a sentença-2 será executada.

Exemplos:

1. Escrever um programa que leia do teclado 2 números inteiros e escreva na tela "O primeiro eh menor" caso o primeiro número fornecido seja menor que o segundo.

```
#include <iostream>
    using namespace std;

main()
{
    int primeiro, segundo;
    cin >> primeiro >> segundo;        // lê os números
```

```
        if(primeiro < segundo)              // compara se o primeiro número é menor
                                            // que o segundo
                cout << "O primeiro eh menor" << endl;     // escreve o texto na tela
                                                          // caso o valor de expressão
                                                          // seja diferente de 0 (zero):
                                                          // verdadeiro (1), no caso, é
                                                          // diferente de 0
}
```

2. Escrever um programa que leia um número inteiro do teclado, verifique se ele está no intervalo de 0 a 10 e, caso sim, diga ao usuário que esta condição foi satisfeita.

```
#include <iostream>
        using namespace std;
main()
{
        int numero;
        cin >> numero;                          // lê o número
        if((numero >= 0) && (numero <= 10))     // faz a verificação especificada
                cout << "Esta no intervalo" << endl;
}
```

3. Dados dois números inteiros, escreva um programa que verifique e exiba na tela mensagens dizendo se eles são iguais ou diferentes.

```
#include <iostream>
        using namespace std;
main()
{
        int primeiro, segundo;
        cout << "Informe dois numeros inteiros: ";
        cin >> primeiro;
        cin >> segundo;

        if (primeiro == segundo)
                 cout >> "São iguais" >> endl;
        else
                cout >> "São Diferentes" >> endl;
}
```

4. Dados três números, fazer um programa que escreva na tela qual é o maior deles.

```
 #include <iostream>
        using namespace std;
main()
{
        int a, b, c;
        cout << “Informe 3 nos. inteiros: “;
        cin >> a >> b >> c;
        if((a > b) && (a > c))
                cout << “O maior eh: ” >> a  << endl;
        else
                if((b > a) && (b > c))
                        cout << “O maior eh: ” >> b  << endl;
                else
                        cout <<“O maior eh: ” << c  << endl;
}
```

Exercício:

Faça um programa que leia do teclado dois números e exiba no monitor de vídeo o maior deles.

7 ESTRUTURAS DE CONTROLE DE REPETIÇÃO

Repetição é uma das ferramentas mais importantes na programação, pois é comum o mesmo trecho de um programa necessitar ser executado repetidas vezes. Isso acontece segundo uma lógica particular de testes.

7.1 *WHILE*()

Uma forma de criar repetições, iterações ou laços em C/C++ é através do comando "*while()*".

Comando *while*() – formato geral

```
while(expressão)
        sentença
```

Funcionamento do comando:

1. Expressão é avaliada, ou seja, tem seu valor calculado e comparado com 0 (zero). Caso o valor seja diferente de 0, vai-se para o passo 2; caso contrário, para o passo 4.
2. Sentença é executada.
3. Desvia-se para o passo 1.
4. Próximo comando, após o *while*().

Exemplo 1:

Escrever um programa que escreva na tela os números pares de 0 a 100.

```
main()
{
        int par;
```

```
        par = 0;
        cout << “Os numeros pares de 0 a 100 sao:” << endl;
        while(par <= 100)
        {
                cout << par << endl;
                par = par + 2;
        }
}
```

Nesse exemplo serão mostrados todos os números pares de 0 a 100. Inicializamos a variável “par” em 0, pois se a utilizarmos sem inicializá-la estaremos manipulando valores desconhecidos.

A linguagem de programação C++ não inicializa variáveis automaticamente, ou seja, se você comparar uma variável não inicializada, essa variável pode possuir qualquer valor (lembra do “*only God knows*” ou “só Deus sabe”?) como por exemplo 1557745, -9524778, 1 ou até mesmo 0.

Outro problema comum é o laço infinito. O *while()* não incrementa ou decrementa automaticamente uma variável; isso deve estar expresso dentro do bloco de instruções, como podemos ver na linha “par = par + 2;”. Caso contrário, “par” sempre seria zero e nunca chegaria a 100 para o laço parar, causando iterações infinitas.

Denominamos a variável “par” de variável de controle. Com o que foi dito, pode-se afirmar que é sempre obrigatória a modificação do conteúdo da variável de controle em um comando *while*(), uma vez que esta sempre deve fazer parte da expressão do *while*().

Problemas envolvendo repetições normalmente podem ser ajustados a dois tipos: aqueles em que conhecemos o número de repetições e aqueles em que o desconhecemos.

No primeiro caso, em que o número de repetições é conhecido, este é informado pelo usuário antes de uma coleção de dados de tamanho definido por aquele número. Neste caso, basta criar-se uma variável de controle para verificar se o tamanho da coleção foi ou não atingido e, enquanto isso não ocorrer, processa-se cada elemento do conjunto. Veja o exemplo a seguir.

Exemplo 2:

Escrever um programa que leia do usuário, via teclado, um número inteiro que identifica a quantidade de temperaturas medidas em uma certa experiência e, a seguir, diversas temperaturas. Estas serão válidas se estiverem no intervalo de 93 a 104 graus da escala Celsius. Ao final, o programa deverá mostrar a média das temperaturas válidas.

```
#include <iostream>
	using namespace std;
main()
{
	int no_temp, i;
	float temp, soma;
	cout << "Qtd de temperaturas: ";
	cin >> no_temp;
	if(no_temp <= 0)
		cout << "Informacao invalida!" << endl;
	else
	{
		i = 0;
		soma = 0;
		while(i < no_temp)
		{
			cout << "Temperatura (" << i + 1 << "): ";
			cin >> temp;
			if((temp >= 93) and (temp <= 104))
			{
				soma = soma + temp;
				i = i + 1;
			}
			else
				cout << “Temperatura invalida!” << endl;
		}
		cout << "Temperatura media = " << soma / no_temp << endl;
	}
}
```

Quando o tamanho do conjunto de dados a processar não é conhecido, normalmente alguma característica dos dados pode ser levada em conta. Para o caso do exemplo anterior, das temperaturas, o final do conjunto pode ser reconhecido por uma temperatura determinada, suponhamos 500°C. O processamento do programa deve, então, ler o primeiro elemento do conjunto antes

de entrar no laço de repetição; o comando while() deverá testar se o final do conjunto foi ou não atingido. Por construção, podemos garantir o não atingimento deste, ou seja, temos o primeiro elemento do conjunto disponível para processamento dentro do laço de repetição. Na sequencia devemos processar um elemento (o primeiro ao entrar no *while*() a primeira vez, e cada um dos demais a cada novo processamento do laço). Ainda por construção, antes de encerrar a repetição, deve-se fazer a leitura de um novo elemento do conjunto; com isso, tem-se a oportunidade de alterar o valor da variável de controle da repetição, que será testada novamente, antes da próxima repetição.

Exemplo 3:

Escrever o mesmo programa do exemplo anterior, porém sem o número de temperaturas informado pelo usuário (caracterizando o conjunto de tamanho desconhecido). O final do conjunto está identificado pela informação da temperatura 500ºC, que não faz parte do mesmo, e é somente utilizada para identificar seu final.

```
#include <iostream>
        using namespace std;
main()
{
        int i;
        float temp, soma;
        i = 0;
        soma = 0;
        cout << “Temperatura (“ << i + 1 << “): “;
        cin >> temp;
        while(temp != 500)
        {
                if((temp >= 93) and (temp <= 104))
                {
                        soma = soma + temp;
                        i = i + 1;
                }
                else
                        cout << “Temperatura invalida!” << endl;
                cout << "Temperatura (" << i + 1 << "): ";
                cin >> temp;
        }
        cout << "Temperatura media = " << soma / i << endl;
}
```

8 FUNÇÕES

O que são funções?

Funções em programação são pequenos programas (subprogramas) que podem ser utilizados (chamados) na função principal, main(), ou em outras funções.

Por que utilizar funções?

É extremamente interessante saber dividir um problema / programa em partes menores, de modo a poder resolvê-lo em pedaços (tarefas) mais simples.

A esta prática de “dividir para conquistar” denomina-se “abordagem *top down*”, ou seja, “de cima para baixo”: nesta abordagem, pensamos qualquer problema como subproblemas, subsubproblemas, e assim por diante, de tal forma que, se resolvidos de forma integrada, resolve-se o problema inicial. Adiante veremos a forma como a linguagem C++ nos permite utilizar esta abordagem. A aplicação desta forma de trabalhar permite dividir as tarefas entre diversos programadores, ou seja, distribuir o processo de programação, fundamental na administração dos recursos humanos envolvidos em projetos de certo porte. Costuma-se denominar cada função (tarefa) identificada de módulo, e à programação decorrente “programação modular”.

O hábito de utilizar funções deixa também os programas mais organizados e, caso ocorram problemas, a tarefa de correção torna-se mais fácil, atuando-se apenas no(s) módulo(s) responsável(is) pelo(s) erro(s).

Outra razão para o uso de funções é para evitar a repetição de código. Ao invés de escrever-se uma sequência de comandos todas as vezes que precisar-se usá-la, é mais inteligente escrever uma função com essas operações e chamá-la quando for necessário.

É interessante ressaltar que você não precisa conhecer o conteúdo de uma função para utilizá-la. Provavelmente nesta altura do curso você se deparou com

algumas funções automáticas da linguagem, como *sqrt*() e *pow*(), ambas da biblioteca "*cmath*", entre outras, que utilizou conhecendo o nome, o que seria colocado entre os parênteses (parâmetros) e o que esperar da saída. A esta característica das funções costuma-se chamar de "conceito caixa preta".

8.1 ESTRUTURA

Funções possuem, como forma geral, um cabeçalho e um corpo, que devem ser codificados após a declaração de constantes de um programa, se houverem, ou, na ausência de constantes declaradas, após a cláusula "using namespace std;", e antes de sua chamada (seu uso), normalmente na função principal "*main*()", como já foi visto anteriormente no tópico "Estrutura geral de um programa em C++".

A forma geral de funções é mostrada, pois, a seguir:

Funções – formato geral

```
Cabeçalho
{
        …
        corpo
        …
}
```

O cabeçalho de uma função tem características estruturais próprias. Através dele, PODE-SE definir (ou não) o uso de áreas de entrada e / ou saída de dados da função, ou seja, por onde vão PODER (ou não) entrar dados para a função e / ou sair dados dela, quando de sua chamada (uso). Entenda-se, neste momento, que a chamada de uma função gera um desvio na sequência de execução dos comandos de um programa, SEMPRE havendo a passagem do controle da execução aos comandos do corpo da função; com isto, queremos esclarecer o porquê das palavras PODE-SE e PODER grafadas com letras maiúsculas – o controle da

execução de comandos sempre passa à função, podendo ou não haver dados de entrada e / ou saída envolvidos.

Cabeçalho de uma função – formato geral

Tipo-de-resultado Nome-da-função([lista de parâmetros])
(retornado por
valor pela função)

Tipo-de-resultado (retornado por valor pela função): este trecho do cabeçalho corresponde a uma área de saída da função, por onde PODEREMOS, eventualmente, devolver ao módulo que chamar a função, um ÚNICO VALOR, do mesmo tipo de dado descrito na codificação da função. Em nosso contexto, por razões minimalistas, trabalhamos apenas com os tipos *int* (inteiro), *float* (real) e *char* (caracter), mas outros tipos de dados que a linguagem suporta também podem ser utilizados. Funções com estes tipos descritos denominam-se “funções tipadas”. Caso optemos (ou não necessitemos), no momento de conceber uma função, por NÃO UTILIZAR esta sua área de saída, deveremos definir seu tipo como *void*. A descrição do tipo de resultado retornado pela função, então, identifica se “controle e um único dado (ou valor)” serão retornados ao módulo que a chamar, por esta área, ao final da execução da função, ou “apenas o controle”. Para as funções tipadas (*int* ou *float* ou *char*), o último comando “executável” no corpo da função deverá ser:

return valor;

onde valor poderá ser um valor propriamente dito (p.e. 0, 1, -2, ´s´, 45.32, …), uma constante, uma variável simples ou uma expressão; **este valor deverá ser compatível com o tipo-de-resultado descrito no cabeçalho da função**.

Nome-da-função: é o nome que resolvemos atribuir-lhe. Costuma-se, ao referir-se a uma função, sufixar-lhe o nome com “()”. Assim, uma função para calcular as raízes de uma equação do segundo grau que resolvemos denominar “req2g” seria descrita como “req2g()”. Para a formação do nome da função utiliza-se as mesmas regras de formação de nomes de constantes e variáveis.

([lista de parâmetros]): componente opcional na escrita de uma função (por isso apresentado entre colchetes, que não devem ser escritos), se usado, este componente permite ativar locais de entrada e/ou locais de entrada e saída de dados da função, ou seja, por onde valores serão informados a ela e / ou recebidos dela. Seu formato completo é:

Lista de parâmetros – formato geral

([tipo[&] parm-1[, tipo[&] parm-2][, tipo[&] parm-3][, …][, tipo[&] parm-n]])
Ou:
([tipo [&] parm-1[, tipo [&] parm-2][, tipo [&] parm-3][, …][, tipo [&] parm-n]])
ou ainda:
([tipo [&]parm-1[, tipo [&]parm-2][, tipo [&]parm-3][, …][, tipo [&]parm-n]])

onde:

tipo[&] parm-i, ou tipo [&] parm-i, ou ainda tipo [&]parm-i, para i = 1, 2, 3, …, n, definem os tipos e nomes de variáveis que serão reconhecidas, na função, como variáveis que conterão dados de entrada somente ou dados de entrada e saída (quando se usa o “&”) da função.

Quando a função for chamada, os valores dos argumentos(*) utilizados na chamada da função serão copiados para os parâmetros correspondentes na lista de parâmetros da função, na mesma posição correspondente ou, como costuma-se dizer, na mesma posição de referência de cada um; assim, se uma função apresentar 5 parâmetros em sua lista de parâmetros do cabeçalho, necessariamente deverão haver 5 argumentos referenciados (ou descritos) na chamada da função.

(*) Adiante, no tópico “chamada de funções”, discorreremos sobre o termo “argumentos”.

Caso utilize-se o “&” junto à descrição de algum parâmetro da função, estaremos declarando que este é de entrada e saída, ou seja:

- ele vai receber um valor (de entrada) do módulo que chamar a função no instante da chamada, como todos os demais que estiverem descritos no cabeçalho da função;
- e, ao final da execução da função, seu valor atual, alterado pela função, será copiado de volta, na mesma posição de referência dele, para o argumento correspondente, naquela posição de referência da lista de argumentos utilizada na chamada da função.

8.2 SAÍDA DE DADOS DE FUNÇÕES

Com o que vimos até este ponto, podemos identificar dois mecanismos de transferência de dados de SAÍDA de funções aos módulos que as chamarem:

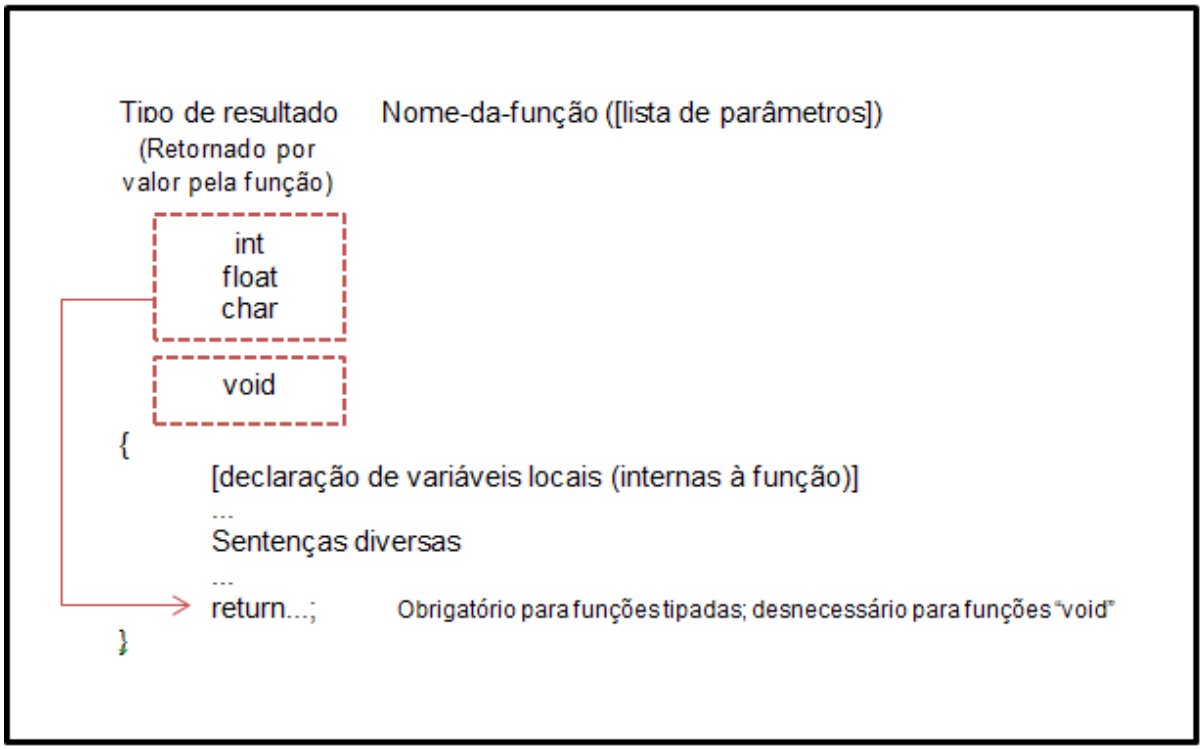
- **retorno por valor**, de valor único, através do uso do comando *return* como último comando executável em funções tipadas;
- e **retorno por referência**, quantos forem necessários, através do uso de parâmetros com “&” na lista de parâmetros da função.

8.3 CORPO DAS FUNÇÕES

Assim como na função principal de qualquer programa, a função *main*(), o corpo das funções é uma sentença composta, delimitada por “{“ e “}”. Da mesma forma que na função *main*(), no corpo das funções descrevemos eventuais declarações de variáveis e demais comandos (atribuições, *if*()s, *while*()s, *cin*()s, *cout*()s, etc., e *return*()). Atente-se para o fato que as variáveis declaradas internamente às funções NÃO SÃO VISÍVEIS pelos módulos que as chamarem; são ditas variáveis locais. Também são locais as eventuais variáveis descritas nas listas de parâmetros das funções, ou seja, só são visíveis do interior das funções onde estiverem declaradas. Ainda que utilize-se o mesmo nome para uma variável externa e internamente a uma função, seus escopos são diferentes, pois os mecanismos de cópia de valores não são desativados; assim, uma variável de

mesmo nome utilizada em funções diferentes irá ocupar locais de memória também diferentes, por isso a importância do conceito de escopo e visibilidade do conteúdo de variáveis. Constantes, por outro lado, são globais a um programa, sendo visíveis não apenas pela função *main*(), mas por qualquer função contida no programa. Para as funções tipadas, o corpo das funções deverá contemplar, como último comando executável, o comando *return*(), obrigatório, pois através deste funciona o mecanismo de retorno por valor deste tipo de funções.

Em resumo, podemos então descrever funções com um esquema estrutural mais completo como a seguir:



8.4 CHAMADA (USO) DE FUNÇÕES

Construímos funções para usá-las. A este ato denominamos chamada das funções. Em decorrência de existirem funções “tipadas” e “não tipadas”, a forma de chamá-las varia:

– para as funções tipadas, sua chamada (uso) é como se fossem expressões pois, ao final de sua execução, acabam transformando-se em um único valor, já que o mecanismo de retorno de dados ao módulo que as chama é, como vimos, por valor; assim, podemos chamar funções deste tipo à direita de atribuições (“=”) ou como expressões em comandos *cout*(), *if*() e *while*();
– já para as funções “não tipadas” ou “tipo void”, a referência a elas deve ser feita como comandos, ou seja, para executá-las escrevemos sua chamada seguida de “;” ou, para ficar bem claro, escrevemos “nome-da-função(...);”.

Ao efetuarmos a chamada de funções, precisamos referir-nos a seus nomes e, correspondentemente a cada parâmetro declarado na lista de parâmetros das funções, devemos enviar valores de entrada a estas, através dos chamados argumentos, coincidentes em número, tipo e posição com aqueles parâmetros nelas declarados, ou seja, o formato geral de fazermos o uso de uma função é:

Chamada de função – formato geral

Nome-da-função([lista de argumentos])
ou:
Nome-da-função([tipo arg-1][, tipo arg-2][, tipo arg-3][, ...][, tipo arg-n]])

onde:

tipo arg-i, para i = 1, 2, 3, … n, definem os tipos e nomes de variáveis (ou apenas valores diretamente informados) reconhecidos, no módulo chamante (onde chamamos determinada função), como os locais onde serão encontrados (ou os valores em si) conteúdos que deverão ser passados à função no momento de sua chamada.

Quando a função for chamada, os valores dos argumentos(*) utilizados na chamada da função serão copiados para os parâmetros correspondentes na lista de parâmetros da função, na mesma posição correspondente ou, como costuma-se dizer, na mesma posição de referência de cada um; assim, como já afirmamos anteriormente, se uma função apresentar 5 parâmetros em sua lista de parâmetros

do cabeçalho, necessariamente deverão haver 5 argumentos referenciados (ou descritos) na chamada da função.

Caso tenha sido utilizado o “&” na descrição de algum parâmetro da função sendo chamada (parâmetro de entrada e saída):

- ele (parâmetro) vai receber um valor (de entrada) do módulo que chamar a função no instante da chamada, como todos os demais que estiverem descritos no cabeçalho da função; este valor é o correspondente do argumento utilizado no módulo que está chamando a função;
- e, ao final da execução da função, o valor do parâmetro, alterado pela função, será copiado de volta, na mesma posição de referência dele, para o argumento correspondente, naquela posição de referência da lista de argumentos utilizada na chamada da função.

8.5 PROTÓTIPOS

Utiliza-se tal construção de programação quando queremos prorrogar a descrição do corpo de uma ou mais funções para após o seu uso, ou seja, queremos chamá-las explorando o conceito de “caixa preta” ou o “desenvolvimento *top down”*.

Protótipos são constituídos apenas dos cabeçalhos das funções seguidos de “;”, sem ser necessário nomear variáveis das listas de parâmetros das funções; apenas declaramos os tipos das variáveis das listas de parâmetros e, se identificada a necessidade de uso de variáveis de entrada e saída nas funções, utilizamos, já nos protótipos, o “&”.

A escrita dos protótipos é feita no mesmo local da estrutura geral de um programa onde escrevemos as funções.

Com este recurso escrevemos, então, a estrutura de nossos programas, utilizando, onde necessário, as funções envolvidas (neste momento apenas protótipos destas) e, após havermos acertado todos os detalhes de interfaces entre

as diversas partes do programa sendo escrito, descrevemos as funções, aí de forma completa, com os nomes de variáveis nas listas de parâmetros das funções e seus corpos.

O formato geral, pois, de um protótipo, é o seguinte:

Protótipo – formato geral

Tipo de resultado Nome-da-função([lista de parâmetros sem nomes de variáveis]);

Com o conceito ora descrito, agora você já sabe porque foi-lhe apresentada a função raiz quadrada como “float sqrt(float)” ou a função potência como “float pow(float, float)”.

8.6 ESPECIFICAÇÃO DE FUNÇÕES

Assim como os programas, as funções devem ser claramente especificadas para que possamos desenvolvê-las.

A especificação completa de uma função deve apresentar:

- o nome da função a desenvolver, na forma nome-da função();
- o objetivo da função, ou seja, uma breve descrição de “o que” deve-se obter de resultado(s) com o uso da função;
- a descrição do(s) algoritmo(s) envolvido(s), ou seja, “como” obter o(s) resultado(s) desejado(s);
- as entradas para a função, ou seja, que valor(es) será(ão) informado(s) à função e por onde (lista de parâmetros ou entrada via teclado, por exemplo);
- as saídas da função, ou seja, o(s) resultado(s) que a função deverá produzir (valor(es) calculado(s), saída escrita em tela, por exemplo);
- um protótipo da função.

8.7 EXEMPLOS

1. Função sem entrada e sem saída

Essa função usa o tipo void(vazio) para dizer que não irá retornar nenhum valor nem vai receber nenhum para sua execução.

```
#include <iostream>
        using namespace std;
// Definição da função verso()
void verso(void)
{
        cout << "Todos verão, o inverso, o inverno." << endl;
}
// Programa Principal
main()
{
        verso();
}
```

Este programa possui 2 funções: verso() e main() (principal). A função criada, “verso()”, imprime via *cout* uma frase. Ela deverá ser declarada antes da *main*(). Para escrever a função verso() após a *main*() poderia haver sido utilizado um protótipo, construção já descrita e que será explorada em maiores detalhes no apêndice deste assunto.

2. Função com retorno de dados por valor

Nome da função: convCF();
Objetivo: converter temperaturas da escala Celsius para a Fahrenheit
Algoritmo envolvido: C / 5 = (F – 32) / 9, sendo C a temperatura na escala Celsius e F na escala Fahrenheit
Entrada (para a função): temperatura na escala Celsius, informada na chamada da função
Saída da função: temperatura na escala Farenheit, enviada ao programa que chamar a função
Protótipo da função: float convCF(float);

Escrever a função especificada e um programa que a utilize; este deverá receber, via teclado, uma temperatura na escala Celsius e, utilizando a função, obter a temperatura equivalente na escala Farenheit e mostrá-la no monitor de vídeo.

```
#include <iostream>
        using namespace std;
float convCF(float C)
{
        float F;
        F = 9. * C / 5. + 32.;
        return F;
}
main()
{
        float tC;
        cout << "Temperatura (Celsius): ";
        cin >> tC;
        cout << tC << " C = " << convCF(tC) << " F" << endl;
}
```

3. Passando resultados por referência

Nome da função: spCF();
Objetivo: calcular a soma e o produto de dois números
Algoritmo envolvido: soma = 1º nº + 2º nº ; produto = 1º nº x 2º nº
Entrada (para a função): 2 números quaisquer
Saída da função: soma e produto dos 2 números
Protótipo da função: void sp(float, float, float&, float&);

Escrever a função especificada e um programa que a utilize; este deverá receber, via teclado, os dois nómeros com os quais deseja-se fazer os cálculos, passá-los à função, receber os resultados e mostrá-los no monitor de vídeo.

```
#include <iostream>
        using namespace std;
void sp(float a, float b, float & s, float & p)
{
        s = a + b;
        p = a * b;
}
main()
{
        float n1, n2, soma, prod;
        cout << "Informe dois nos. quaisquer: ";
        cin >> n1 >> n2;
        sp(n1, n2, soma, prod);
        cout << "(" << n1 << ") + (" << n2 << ") = " << soma << endl;
        cout << "(" << n1 << ") * (" << n2 << ") = " << prod << endl;
}
```

4. Usando retorno de dados por valor e por referência

Nome da função: raizq();
Objetivo: calcular a raiz quadrada de um número real
Algoritmo envolvido:
- receber um nº real qualquer na área de entrada de dados da função;
- validá-lo, pois não podemos calcular a raiz quadrada de números negativos;
  - se for possível o cálculo, passar esta informação ao programa que utilizar a função, em conjunto com a raiz quadrada obtida;
  - caso o cálculo não seja possível, passar APENAS esta informação ao programa que utilizar a função

Entrada (para a função): 1 número real quaisquer
Saída da função:
- 0: cálculo não possível;
- 1 e a raiz quadrada do nº recebido

Protótipo da função: int raizq(float, float&);

Escrever a função descrita e um programa que a utilize.

```
#include <iostream>
#include <cmath>
        using namespace std;
int raizq(float x, float &rx)
{
        if(x < 0)
                return 0;
        rx = sqrt(x);
        return 1;
}
main()
{
        float no, r;
        cout << "No.: ";
        cin >> no;
        if(raizq(no,r))
                cout << "Raiz quadrada de " << no << " = " << r << endl;
        else
                cout << "Numero invalido!" << endl;
}
```

5. Um exemplo ilustrando tudo o que foi visto

```
#include <iostream>
        using namespace std;
```

```
void apresentar_se();
int ler_lim_i_s();
int validar_lims(int, int);
int eh_primo(int);
void listar_primos(int);
void contar_nos(int, int&, int&);
void listar_totais(int, int, int, int);
main()
{
	int li, ls, ni, tnp, tp;
	apresentar_se();
	cout << "Limite inferior: ";
	li = ler_lim_i_s();
	cout << "Limite superior: ";
	ls = ler_lim_i_s();
	if(validar_lims(li,ls) == 0)
		cout << "Limites invalidos!" << endl;
	else
	{
		cout << endl;
		cout << "Numeros primos em [" << li << "," << ls << "]:" << endl;
		ni = li;
		tnp = 0;
		tp = 0;
		while(ni <= ls)
		{
			if(eh_primo(ni))
			{
				listar_primos(ni);
				contar_nos(1,tnp,tp);
			}
			else
				contar_nos(0,tnp,tp);
			ni = ni + 1;
		}
		listar_totais(li,ls,tnp,tp);
	}
}
void apresentar_se()
{
	cout << "Programa para listar numeros primos entre dois limites inteiros," <<
endl;
	cout << "crescentes, positivos e nao nulos informados!" << endl;
	cout << endl;
}
int ler_lim_i_s()
{
	int no;
```

```
	cin >> no;
	return no;
}
int validar_lims(int lim_i, int lim_s)
{
	if((lim_i <= 0) or (lim_s <= 0) or (lim_i > lim_s))
	return 0;
	return 1;
}
int eh_primo(int no)
{
	int d, qd;
	if((no == 1) or (no == 2) or (no ==3))
		return 1;
	d = 2;
	qd = 0;
	while(d <= (no / 2))
	{
		if((no % d) == 0)
			qd = qd + 1;
		d = d + 1;
	}
	if(qd)
		return 0;
	return 1;
}
void listar_primos(int no)
{
	cout << no << " ";
}
void contar_nos(int indicador, int& qnp, int& qp)
{
	if(indicador)
		qp = qp + 1;
	else
		qnp = qnp + 1;
}
void listar_totais(int lim_i, int lim_s, int qnp, int qp)
{
	cout << endl << endl;
	cout << "[" << lim_i << "," << lim_s << "]: " << qnp + qp << " numeros
processados" << endl;
	cout << "Primos: " << qp << "   Nao primos: " << qnp << endl;
}
```

9 VETORES E MATRIZES

Vetores e matrizes, genericamente denominados “*arrays*”, são estruturas de dados que podem nos ajudar muito quando temos muitas variáveis de um mesmo tipo e precisamos guardar seus conteúdos originais para uso posterior.

Imagine o seguinte problema: você precisa criar um algoritmo que lê os números e as 4 notas de 50 alunos, calcule a média de cada aluno e informe quais foram aprovados, quais foram reprovados e quais tiveram média igual ou superior à da turma. Percebe-se que muitas variáveis serão necessárias para isso. Façamos as contas: 50 variáveis para armazenar os códigos (números) dos alunos, 200 variáveis para armazenar as 4 notas de cada aluno (4*50), e por fim, 50 variáveis para armazenar as médias de cada aluno, correspondendo a 300 variáveis no total, isso sem falar da quantidade de linhas de código do programa para ler todos os dados do usuário, calcular as médias e apresentar os resultados. Ademais, para podermos dizer que um aluno ficou acima ou abaixo da média da turma, precisamos ter todas as médias individuais guardadas, não podendo utilizar as soluções de leitura destrutiva até então utilizadas, reutilizando variáveis a cada ciclo de entrada de dados.

Felizmente tarefas desse tipo podem ser simplificadas com o uso de *arrays*, em especial vetores e matrizes. Para tanto, precisamos conhecer melhor essas estruturas.

Array é o termo geral usado para designar conjuntos (coleções) de objetos de um mesmo tipo.

9.1 VETORES

Um vetor é um conjunto unidimensional de variáveis do mesmo tipo (int, float, char, ...).

De forma mais apurada e afim com seu uso em informática, podemos definir um vetor v, de tamanho N, ou seja, com N elementos de um mesmo tipo, como:

$$(v)_N = \{\text{elementos } v_i \mid v_i \text{ Ɛ mesmo tipo, com } i = 0, 1, 2, \ldots, i-1, i, i+1, \ldots, N-1\}.$$

Sua declaração é feita da seguinte maneira:

Declaração de vetores – formato geral

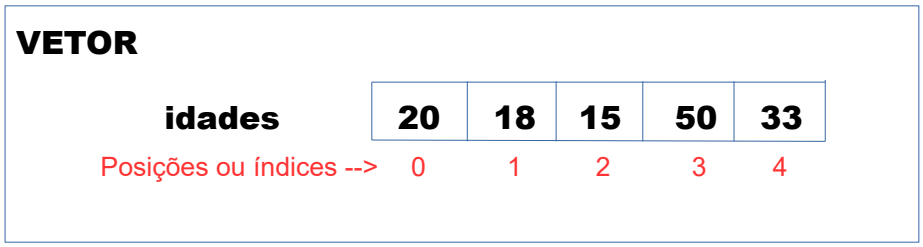
tipo nome-do-vetor[tamanho], ...;

Por exemplo, consideremos a declaração:

int idades[5];

Esta declara um vetor de nome “idades”, com 5 elementos (tamanho do vetor) ou posições. O número de elementos de um vetor é um número inteiro e constante, podendo ser descrito literalmente (como neste exemplo) ou através de uma constante inteira, estabelecida pela diretiva “#define”.

Imaginemos o vetor declarado acima como o ilustrado abaixo:



Os contéudos dos elementos do vetor são acessados pelo nome do vetor e um índice inteiro, entre colchetes ([índice]). O primeiro elemento está na posição 0, ou seja, idades[0], com valor 20, o segundo, na posição 1, idades[1], com valor 18, e assim sucessivamente, até a posição tamanho-1, neste caso o elemento idades[4], com conteúdo 33.

CUIDADO: as linguagens C e C++ não verificam se você está utilizando um índice inválido, dessa forma, você pode acabar lendo “lixo” ou alterando outras variáveis na memória se não controlar de forma eficaz o índice utilizado.

Exemplo: encontrar o maior valor dentro de um vetor.

```
#include <iostream>
        using namespace std;
#define N 10
main()
{
```

```
        int i, vetor[N], maior;
        cout << “Informe o vetor (“ << N << “ elementos):” << endl;
        i = 0;
        while(i < N)
        {
                cin >> vetor[i];                // lê os dados
                i = i + 1;
        }
        maior = vetor[0];                       // assume que o primeiro elemento é o
                                                // maior
        i = 1;                                  // o primeiro elemento, com índice zero,
                                                // não precisa ser testado
        while(i < N)
        {
        if(vetor[i] > maior)
                maior = vetor[i];
                i = i +1;
        }
        cout << “Maior elemento: “ << maior << endl;
}
```

Resultado da execução do programa:

```
Informe o vetor (10 elementos):
25 13 15 22 37 44 12 23 29 32
Maior elemento: 44

Process returned 0 (0x0)   execution time : 80.584 s
Press any key to continue.
```

O processamento dos dados (conteúdos) de um vetor deve sempre tratar todos os seus elementos, um a um. Assim, este tratamento estará normalmente atrelado ao uso de um comando de repetição. O exemplo apresentado acima ilustra esta forma de trabalhar.

Assim como as variáveis simples (aquelas que não possuem índice associado), vetores, após definidos, têm seus elementos com valores (conteúdos) desconhecidos: um “lixo” qualquer (“oGk” ou “sDs”). Portanto, é necessário “inicializar” (colocar valores iniciais, de nosso interesse) nos elementos de um vetor. Como para as variáveis simples, há duas formas de fazer isto, ou seja, de alterar os conteúdos dos elementos de um vetor:

– via entrada de dados pelo teclado, com o comando:
  cin >> nome-do-vetor[índice] >> ...;

ou:

– através de atribuição:
  nome-do-vetor[índice] = …;

Uma vez inicializado (carregado / alterado) um vetor, podemos então passar a processá-lo. No exemplo apresentado, a carga do vetor foi feita via leitura de dados do teclado e, após, passou-se a processá-lo (acessar / recuperar seus elementos) para identificar o maior conteúdo armazenado no vetor. Este acesso pode ser feito das seguintes formas:

– referindo-se, em um comando “cout”, a um elemento do vetor (nome-do-vetor[índice]) diretamente ou como parte de uma expressão aritmética, relacional ou lógica;

  ou:

– referindo-se a um elemento do vetor (nome-do-vetor[índice]) como parte de uma expressão aritmética, relacional ou lógica à direita de um comando de atribuição (variável = … nome-do-vetor[índice] ...). No exemplo apresentado, na segunda repetição, o comando “maior = vetor[i];” ilustra essa forma de acesso aos elementos do vetor;

  ou ainda:

– referindo-se a um elemento do vetor (nome-do-vetor[índice]) como parte de uma expressão aritmética, relacional ou lógica de um comando “if(...)” ou “while(...)”. No exemplo apresentado, na segunda repetição, a linha onde está escrito “if(vetor[i] > maior)” ilustra uma dessas formas de acesso aos elementos de um vetor.

9.1.1 FUNÇÕES ENVOLVENDO VETORES

O processamento de vetores utilizando funções é também suportado pela linguagem C++, bastando declarar-se o(s) vetor(es) envolvido(s) na lista de parâmetros do cabeçalho das funções. O mecanismo de transferência de dados entre os módulos (funções) chamantes e chamados, no entanto, para vetores, é SEMPRE “por referência”, não sendo necessário o uso de “&” para que isto aconteça. Com isso, os vetores envolvidos em funções são entendidos SEMPRE como variáveis de saída por aquelas, ou seja, toda(s) e qual(is)quer alteração(ões) feitas por uma função em um vetor são devolvidas ao módulo chamante ao final da execução da função.

Exemplo: Desenvolver um programa em C++ para ler as temperaturas observadas em um experimento, calcular a temperatura média e o desvio médio das temperaturas registradas. Foram medidas 8 temperaturas ao longo do experimento. Para o cálculo do desvio médio, deve-se apurar a média das diferenças, em módulo, entre as temperaturas registradas e a média aritmética calculada das temperaturas.

```
#include <iostream>
#include <cmath>
        using namespace std;
#define N 8
void ler_temperaturas(float []);
float calc_media_temperaturas(float []);
float calc_dm_temperaturas(float, float []);
void mostrar_resultados(float, float);
main()
{
        float T[N];
        ler_temperaturas(T);
        mostrar_resultados(calc_media_temperaturas(T),
                                calc_dm_temperaturas(calc_media_temperaturas(T), T));
}
void ler_temperaturas(float t[])
{
        int i;
        cout << "Temperaturas registradas:" << endl;
        i = 0;
        while(i < N)
        {
                cout << "[" << i + 1 << "]: ";
                cin >> t[i];
                i = i + 1;
        }
}
float calc_media_temperaturas(float t[])
{
        float s;
        int i;
        i = 0;
        s = 0;
        while(i < N)
        {
                s = s + t[i];
                i = i + 1;
        }
        return s / N;
}
float calc_dm_temperaturas(float t_media, float t[])
{
        float s;
        int i;
        i = 0;
        s = 0;
        while(i < N)
        {
                s = s + fabs(t[i] - t_media);
                i = i + 1;
```

```
        }
        return s / N;
}
void mostrar_resultados(float t_md, float t_dm)
{
        cout << "Temperatura media: " << t_md << endl;
        cout << "Desvio medio: " << t_dm << endl;
}
```

Observar, no programa acima, que o vetor de temperaturas (T) teve sua definição de tamanho declarada apenas em um único ponto do programa, na função main(). Isto é possível, e permitido na linguagem, pelo conceito de ponteiros (endereços de memória) associados ao conceito de variável, ou seja, à correspondência biunívoca existente entre o nome de uma variável e seu endereço na memória. Como vetores também são variáveis, porém indexadas, basta ao sistema conhecer o endereço inicial de memória onde o vetor se encontra para que este possa calcular, a partir do produto da quantidade de bytes associada ao tipo do vetor e seu tamanho (número de elementos), o espaço de memória ocupado pelo vetor, bem como qualquer elemento que precise ser tratado.

O resultado da execução do programa está ilustrado abaixo:

```
Temperaturas registradas:
[1]: 98
[2]: 98.5
[3]: 99
[4]: 99.5
[5]: 100
[6]: 101
[7]: 101.5
[8]: 102.5
Temperatura media: 100
Desvio medio: 1.25

Process returned 0 (0x0)   execution time : 66.766 s
Press any key to continue.
```

9.1.2 PROBLEMAS TÍPICOS ENVOLVENDO VETORES

- cálculo de medidas de tendência central e de dispersão dos elementos de um vetor (último exemplo apresentado);
- expressões com vetores, multiplicação de escalar por vetor e soma algébrica de vetores;

- busca / tratamento de elementos de um vetor (e / ou índices) sujeitos a determinadas condições;
- produto escalar (ou interno) de vetores;
- tratamento da diferenciação de conceitos de tamanhos físico e lógico de vetores;
- concatenação de vetores;
- intercalação de vetores;
- inversão de vetores;
- ordenação de vetores.

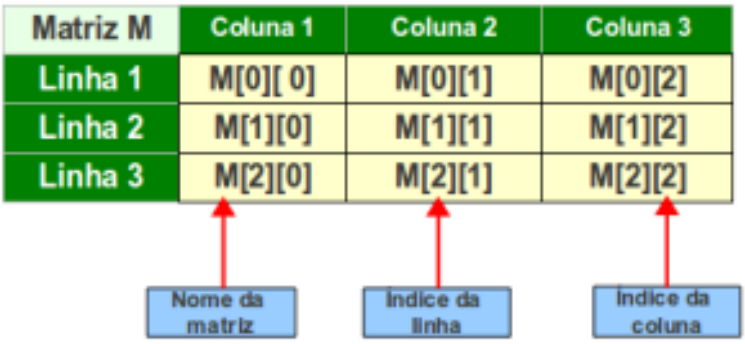
9.2 MATRIZES

Matriz é uma estrutura de dados do tipo array com duas ou mais dimensões. Assim como nos vetores, os elementos de uma matriz também têm todos o mesmo tipo de dado (*int*, *float*, *char*, …). Não é à toa que são chamadas de “vetores de vetores”. Na prática, as matrizes formam tabelas na memória e, se possuírem duas dimensões ou índices, devem ser declaradas assim:

Declaração de matrizes – formato geral

tipo nome-da-matriz[número_de_linhas][número_de_colunas], ...;

Exemplo: declaração, colocação (carga) de valores em uma matriz quadrada 3x3 (3 linhas e 3 colunas) e exibição de alguns valores.



```
#include <iostream>
        using namespace std;
main()
{
        int M[3][3];                    // declaração da matriz M
        M[0][0] = 4;                    // colocação (carga), por atribuição,
        M[0][1] = 10;                   //      de valores nos elementos da matriz,
        M[0][2] = 403;                  //      por linha
        M[1][0] = 9;                    // Notar que fixou-se o índice de linha
        M[1][1] = 33;                   //      (primeiro índice) com o mesmo valor de
        M[1][2] = 584;                  //      3 em 3 elementos, enquanto o índice de
        M[2][0] = 1;                    //      coluna (segundo índice) teve seus valores
        M[2][1] = 11;                   //      variando, dentro do ranking de valores
        M[2][2] = 100;                  //      possíveis para o mesmo
        cout << M[0][1] << endl;        // imprime o valor 10 na tela do usuário
        cout << M[2][2] << endl;        // imprime o valor 100
}
```

Outro exemplo: carga e mostra de uma matriz de números inteiros de três linhas e quatro colunas.

```
#include <iostream>
        using namespace std;
#define L 3
#define C 4
main()
{
        int X[L][C], i, j;
         /* Carga da matriz com valores digitados pelo usuário */
        cout    << “Informe matriz (“ << L << “ linhas e “ << C << “ colunas), por linha:”
                << endl;
        i = 0;
        while(i < L)
        {
                j = 0;
                while(j < C)
                {
                        cin >> X[i][j];
                        j = j + 1;
                }
                i = i + 1;
        }
        /* Mostra da matriz */
        cout << “Matriz informada (“ << L << “ x “ << C << “):” << endl;
        i = 0;
        while(i < L)
        {
                j = 0;
```

```
            while(j < C)
            {
                    cout << X[i][j] << " ";
                    j = j + 1;
            }
            cout << endl;
            i = i + 1;
        }
}
```

Resultado da execução do programa:

```
Informe matriz (3 linhas e 4 colunas), por linha:
10 11 12 13
14 15 16 17
25 24 19 20
Matriz informada (3 x 4):
10 11 12 13
14 15 16 17
25 24 19 20

Process returned 0 (0x0)   execution time : 543.302 s
Press any key to continue.
```

9.2.1 EXTENSÃO DE CONCEITOS DE VETORES A MATRIZES

Tudo aquilo que foi afirmado e mostrado relativamente a vetores aplica-se integralmente às matrizes, respeitada a diferença de estarmos agora tratando de duas dimensões (índices) e não mais de apenas uma.

Assim, a partir da definição de vetores apresentada, podemos esboçar uma definição de uma matriz M, com d1 linhas e d2 colunas, ou d1 x d2 elementos de um mesmo tipo, como:

$$(M)_{d1 \times d2} = \{\text{elementos } M_{ij} \mid M_{ij} \in \text{mesmo tipo},$$
$$\text{com } i = 0, 1, 2, \ldots, i-1, i, i+1, \ldots, d1 - 1,$$
$$\text{e } j = 0, 1, 2, \ldots, j-1, j, j+1, \ldots, d2 - 1\}$$

O processamento dos dados (conteúdos) de uma matriz deve, assim como nos vetores, sempre tratar todos os seus elementos, um a um. Como agora estamos tratando de uma estrutura de dados com duas dimensões, este tratamento estará normalmente atrelado ao uso de dois comandos de repetição aninhados, um para tratar cada dimensão. O último exemplo, apresentado acima, ilustra esta forma de trabalhar.

Inicialização (processo de carga / alteração) e processamento (acesso / recuperação) de valores dos elementos de uma matriz também estão sujeitos às mesmas regras seguidas para vetores.

O processamento de matrizes envolvidas em funções também segue as mesmas restrições descritas para vetores.

Uma pequena diferença prática ao processar-se matrizes utilizando funções observa-se na declaração de suas dimensões. No processamento de vetores com funções, como vimos, a declaração do tamanho de um certo vetor compartilhado entre funções pode ser feita em um único ponto, face à associação variável x ponteiro (endereço de memória) existente. Ao processar-se matrizes, esta regra continua valendo para a primeira dimensão da matriz apenas, não para as demais, com declaração sempre obrigatória. Na prática, então, costuma-se declarar todas as dimensões das matrizes envolvidas.

9.2.2 PROBLEMAS TÍPICOS ENVOLVENDO MATRIZES

Um primeiro grupo de problemas atrelados ao processamento de matrizes desenvolve-se relativamente às suas dimensões.

Uma matriz com dimensões iguais em número de linhas e de colunas diz-se “quadrada”. Propriedades típicas das matrizes quadradas, em uma matriz quadrada N x N, são apresentadas a seguir.

Seja a matriz X[NxN]:

$$(X)_{N \times N} = \begin{pmatrix} X_{00} & X_{01} & X_{02} & \dots & X_{0\,N-1} \\ X_{10} & X_{11} & X_{12} & \dots & X_{1\,N-1} \\ X_{20} & X_{21} & X_{22} & \dots & X_{2\,N-1} \\ \dots & \dots & \dots & \dots & \dots \\ X_{N-1\,0} & X_{N-1\,1} & X_{N-1\,2} & \dots & X_{N-1\,N-1} \end{pmatrix}$$

Considerados os índices “i” para as suas linhas e “j” para as colunas, temos as propriedades:

- diagonal principal: i = j;
- diagonal secundária: (i + j) = (N – 1);
- triângulo principal superior (t.p.s.): i < j;
- triângulo principal inferior (t.p.i.): i > j;

- triângulo secundário superior (t.s.s.): (i + j) < (N – 1);
- triângulo secundário inferior (t.s.i.): (i + j) > (N – 1).

Obter somas, diferenças, produtos ou transformações (alterações) de elementos com índices que atendem a alguma(s) destas propriedades são problemas típicos.

Considerando as matrizes de forma geral, em qualquer matriz que possua “m” linhas (m ≥ 2) e “n” colunas (n ≥ 2), é sempre possível identificar:

- cantos da matriz: elementos com índices (i,j) = (0,0) ou (i,j) = (0,n-1) ou (i,j) = (m-1,0) ou (i,j) = (m-1,n-1);
- bordas da matriz: elementos com índice de linha i = 0 ou i = m-1, ou índice de coluna j = 0 ou j = n-1.

Caso a matriz “m x n” tenha “m” ≥ 3 linhas e “n” ≥ 3 colunas, surge uma nova figura ou propriedade a tratar: o “miolo” da matriz, ou seja, aqueles elementos que não se situam em suas linhas ou colunas externas, ou que não pertencem às suas bordas.

Outros problemas:

- obtenção de matriz transposta;
- tratamento de submatrizes;
- relativos ao tratamento de linhas ou colunas de matrizes como vetores;
- tombamento ou giro de matrizes a 90^0, 180^0 e 270^0;
- tratamento de matrizes segundo definições e demandas da álgebra linear, engenharias e métodos numéricos, entre outros, como produto, inversão de matrizes, registros e cálculos de topografia, etc.

APÊNDICE 1 – TABELA ASCII

Antes de 1960, a maioria dos sistemas de computadores tinha uma maneira particular de representar os caracteres alfanuméricos. Ou seja, cada marca de computador, e até cada modelo, usava um código próprio para representar cada letra, número e todos os símbolos que usamos.

Assim, foi proposto o uso de um código comum, utilizado por todos, a fim de possibilitar a comunicação entre os computadores, objetivando permitir a troca de dados entre máquinas de diferentes tipos e fabricantes.

O *"American National Standards Institute"* aceitou a proposta de Robert William Bemer e sua equipe para desenvolver o *"American Standard Code for Information Interchange"* (Código Padrão Americano para Intercâmbio de Informações), hoje conhecido como código ASCII.

Portanto, ASCII é um código numérico usado para representar os caracteres, entendido por quase todos os computadores, impressoras e programas de edição de texto.

Siga o *link* indicado na referência abaixo para saber mais sobre o assunto “representação de caracteres”.

Referência:

Código ASCII. Disponível em: <http://cultura.ufpa.br/dicas/progra/arq-asc.htm>. Acesso em 30 de agosto de 2016.

APÊNDICE 2 – MÁQUINA VIRTUAL

Diferença entre Dados e Endereços

Vamos supor que você more na Rua Albertina, número 254. Temos então que o seu endereço é representado por Rua Albertina – 254 e o conteúdo deste endereço é sua casa. A tabela abaixo ilustra a situação, assim como as outras entidades que então em torno de você:

Endereço de Memória	Conteúdo
Rua Albertina – 254	Sua Casa
Rua Albertina – 255	Seu Vizinho
Rua Albertina – 256	Panificadora do João

De forma análoga isso ocorre no computador, como por exemplo:

Endereço de Memória	Conteúdo
1024	323

Nesse caso, o endereço de memória 1024 guarda o conteúdo 323.

Cada dado possui, portanto, um espaço de memória reservado a ele e, para acessá-lo, é necessário saber seu endereço. Caso o computador tente acessar um espaço de memória por meio do endereço 1024, o retorno será 323.

Ciclo de Execução e Instruções

Considere o seguinte ciclo de execução, na qual utilizaremos a notação [ ] para endereço e p para o conteúdo.

Tabela 1: Instruções

Código	Mnemônico	Descrição	Notação
1	load	escreva em [p+1] o valor do número em [p+2] e some 3 em p	[p+1] ← [p+2]
2	add	escreva em [p+1] o valor da subtração do número em [[p+2]] pelo número em [[p+3]] e some 4 em p	[p+1] ← [[p+2]] + [[p+3]]
3	sub	escreva em [p+1] o valor da subtração do número em [[p+2]] pelo número em [[p+3]] e some 4 em p	[p+1] ← [[p+2]] - [[p+3]]
4	mult	escreva em [p+1] o valor do produto dos números em [[p+1]] e [[p+3]] e some 4 em p	[p+1] ← [[p+2]] x [[p+3]]
5	div	escreva em [p+1] o valor do divisão dos números em [[p+1]] e [[p+3]] e some 4 em p	[p+1] ← [[p+2]] / [[p+3]]
6	sqrt	escreva em [p+1] o valor da raiz quadrada de [[p+2]] e some 3 em p	[p+1] ← raiz quadrada de [[p+2]]
7	read	leia um número do teclado, escreva-o em [p+1] e some 2 em p	[p+1] ←Entrada
8	write	escreva [[p+1]] na tela e some 2 em p	Escreva ← [[p+1]]
9	stop	pare	

Considere a tabela inicial de endereços e seus respectivos conteúdos:

Tabela 2: Inicial

Endereço	Conteúdo
0	1
1	20
2	3
3	1
4	27
5	2
6	7
7	90
8	2
9	60
10	4
11	1

1 - Programa começa com p = 0.

2 - Agora é preciso interpretar [p] = [0] = 1. Como o conteúdo é 1, temos pela tabela que ele é um LOAD com a descrição: “escreva em [p+1] o valor do número em [p+2] e some 3 em p”. Então escreva em [0+1] o valor de [0+2], ou seja, o endereço 1 vai receber o que havia no endereço 2, e p vale 3.

Endereço	Conteúdo
0	1
1	3
2	3

3 - Como p = 3, vamos interpretar que [3] = 1. Consequentemente, teremos mais um LOAD a ser executado: “escreva em [p+1] o valor do número de [p+2] e some 3 em p”. Então escreva em [3+1] o valor de [3+2] e p = 3 + 3 = 6.

……..

3	1
4	2
5	2

…….

4 - Como p = 6, vamos interpretar [6] = 7. Teremos então um READ a ser executado, e pela tabela segue a seguinte descrição: “espere usuário digitar algo no teclado, e carregue este valor em [p+1] e some 2 em p”. Assim, o valor que tinha em [p+1] = [7] será substituído pelo que o usuário entrou no teclado e p valerá 6+2 = 8.

……..

6	7
7	Primeira Entrada do Usuário

…….

5 - Como p = 8 vamos interpretar [8] = 2. Assim, teremos um ADD, ou seja, uma soma a ser executada, com a seguinte descrição: “escreva em [p+1] o valor da subtração do número em [[p+2]] pelo número em [[p+3]] e some 4 em p”. Temos:

[p+1] = [8+1] = [9]

[[p+2]] = [[8+2]] = [[10]] e como [10] tem o conteúdo 4, temos que [4] = 2

[[p+3]] = [[8+3]] = [[11]] e como [11] tem o conteúdo 1, temos [1] = 3

Logo, em [9] será guardada a soma de 2 + 3.

……..

8	2
9	5
10	4
11	1

…….

E o programa vai seguindo instruções até encontrar o código 9, que seria o STOP.

Após estas execuções, a tabela tomará a seguinte forma:

0	1
1	3
2	3
3	1
4	2
5	2
6	7

7	Primeira Entrada do Usuário
8	2
9	5
10	4
11	1

Aumentando a Abstração

A partir da Tabela 2, podemos fazer uma abordagem diferente, separando em colunas: o endereço, instrução, operando 1, operando 2 e operando 3.

Endereço	Instrução	Operando 1	Operando 2	Operando 3
0	1	20	3	
3	1	27	2	
6	7	90		
8	2	60	4	1

Utilizando esta abordagem, percebe-se que basta ver o que cada número de 1 a 9 significa e executar suas respectivas instruções. Sendo assim, não precisamos olhar para o endereço e sim apenas para as instruções. Portanto, podemos reescrevê-lo da seguinte forma:

Instrução	Operando 1	Operando 2	Operando 3
1	20	3	
1	27	2	

7	90		
2	60	4	1

E substituindo o número da instrução pelo mnemônico da instrução, temos:

Instrução	Operando 1	Operando 2	Operando 3
LOAD	20	3	
LOAD	27	2	
READ	90		
ADD	60	4	1

E se substituirmos os operandos por variáveis genéricas, teremos:

Instrução	Operando 1	Operando 2	Operando 3
LOAD	A	3	
LOAD	B	2	
READ	C		
ADD	D	A	B

Agora deixando em notação das instruções:

Instrução	Notação
LOAD	A ← 3

LOAD	B ← 2
READ	C <- Recebe a primeira entrada do usuário
ADD	D ← A + B

Com isso temos que podemos sair do baixo nível computacional, e aumentar essa abstração aos poucos, até chegar o momento em que trabalhamos apenas com instruções e variáveis genéricas.

Criando o Primeiro Programa

Entendido o conjunto de comandos, o endereço e os conteúdos na memória, vamos criar nosso primeiro programa. Vamos supor que o programa deva ler do usuário a soma e a diferença de dois valores e exibir na tela quais são esses dois valores.

O esquema matemático seria:

$$\begin{cases} a+b=x \\ a-b=y \end{cases} \quad (+)$$

$$2a = x + y \qquad b = x - a$$

$$a = \frac{x+y}{2} \qquad b = x - \frac{x+y}{2}$$

Passando para a linguagem de programação, teremos:

$$\begin{cases} \text{valor_1} + \text{valor_2} = \text{soma} \\ \text{valor_1} - \text{valor_2} = \text{diferença} \end{cases}$$

Endereço na memória	Nome da variável
34	soma
25	diferença

40	soma_mais_diferença
33	meio_soma_mais_diferença
52	valor_1
39	valor_2
57	dois

Fazendo as atribuições dos valores às variáveis:

soma ← (entrada do usuário)
diferença ← (entrada do usuário)
dois ← 2*
soma_mais_diferença ← soma + diferença
meio_soma_mais_diferença ← $\frac{\text{soma_mais_diferença}}{dois}$
valor_1 ← meio_soma_mais_diferença
valor_2 ← soma – meio_soma_mais_diferença
● ← valor_1**
● ← valor_2
*A variável dois receberá o valor do número inteiro 2
** ● significa impressão na tela

Formas semelhantes de se escrever esse programa:

soma ← (entrada do usuário)
diferença ← (entrada do usuário)
soma_mais_diferença ← soma + diferença
meio_soma_mais_diferença ← $\frac{\text{soma_mais_diferença}}{dois}$
valor_1 ← meio_soma_mais_diferença
valor_2 ← soma – valor_1
● ← valor_1
● ← valor_2

read soma
read diferença
soma_mais_diferença ← soma + diferença
valor_1 ← $\frac{\text{soma_mais_diferença}}{dois}$
valor_2 ← soma – valor_1
write valor_1

write valor_2

read soma
read diferença
valor_1 ← $\frac{\text{soma_mais_diferença}}{dois}$
valor_2 ← soma – valor_1
write valor_1
write valor_2

Até que chegamos em C++:

```
#include <iostream>
        using namespace std;
main ( )
{
                float soma, diferença;
                cin>>soma;
                cin>>diferença;
                cout<<(soma+diferença)/2<<endl;
                cout<<soma-((soma+diferença)/2)<<endl;
}
```

Referência

CASTILHO, M.; SILVA, F.; WEINGAERTNER, D. **Algoritmos e Estrutura de Dados I**. Agosto de 2015.

APÊNDICE 3 – TIPOS DE DADOS

A tabela abaixo mostra todos os tipos de dados básicos definidos no padrão ANSI.

Tipo	Tamanho em Bytes	Faixa de valores
char	1	-127 a 127
unsigned char	1	0 a 255
signed char	1	-127 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 dígitos de precisão
double	8	Dez dígitos de precisão
long double	10	Dez dígitos de precisão

Referência:

Tipos de Dados, Constantes e Variáveis. Disponível em: <http://equipe.nce.ufrj.br/adriano/c/apostila/tipos.htm>. Acesso em 30 de agosto de 2016.

APÊNDICE 4 – OPERADORES DE ATRIBUIÇÃO ALTERNATIVOS

Este apêndice apresenta um conjunto complementar de operadores de atribuição disponíveis nas linguagens C / C++.

OPERADOR	SINTAXE	SIGNIFICADO
Incremento pós-fixado	A++	A = A + 1
Decremento pós-fixado	A--	A = A - 1
Atribuição por adição	A += B	A = A + B
Atribuição por subtração	A -= B	A = A - B
Atribuição por multiplicação	A *= B	A = A * B
Atribuição por divisão	A /= B	A = A / B;
Atribuição por resto	A %= B	A = A % B

Fontes:

DELGADO, A. L. N; **Linguagem C++ - Notas de Aula.** Curitiba, junho de 2012.

Operadores em C e C++. Disponível em: <https://pt.wikipedia.org/wiki/Operadores_em_C_e_C%2B%2B>. Acesso em 30 de agosto de 2016

APÊNDICE 5 – PRECEDÊNCIA DE OPERADORES

A seguir, a precedência dos operadores utilizados em expressões, em ordem decrescente. Elementos da mesma linha terão prioridade da esquerda para a direita na sentença que estiver escrita no programa.

OPERADOR
( ) [ ]
++ --
* / %
+ -
<= < >= >
== !=
&&
||
+= -= *= /= %=

APÊNDICE 6 – ESTRUTURA DE REPETIÇÃO *FOR*

Alternativamente ao comando de repetição *while()*, as linguagens C / C++ disponibilizam outro comando de repetição: o *for()*, utilizado quando queremos executar um bloco de códigos um determinado número de vezes.

Este comando é composto de um cabeçalho, dividido em três partes, seguido de uma sentença, com o formato abaixo:

for(início; condição; incremento)
sentença

A parte “início” é executada uma única vez (na entrada do laço). Geralmente é nessa parte que declaramos e inicializamos a variável de controle. Em “condição” testa-se a situação atual da variável de controle, para verificar se o laço deve ou não continuar sua execução. Já na parte do “incremento” altera-se o valor da variável de controle. Por fim, a sentença (simples ou composta) representa o “bloco de instruções” que deverão ser executadas repetidas vezes.

Exemplo:

```
#include <iostream>
        using namespace std;
main()
{
        // um laço for que mostra os inteiros  de 10 a 0
        for(int i = 10; i >= 0; i = i - 1)
        {
                cout << i << endl;
        }
}
```

EXERCÍCIOS: 1 – EXPRESSÕES

1) * (progarit) Dados o primeiro termo e a razão de uma progressão aritmética, determinar a soma dos seus primeiros cinco termos.

As fórmulas que definem os termos e soma de uma progressão aritmética (PA) são:

$$a_n = a_1 + (n-1) \times r$$
$$S = \frac{a_1 + a_n}{2} \times n$$

```
#include <iostream>
using namespace std;

int main()
{
	int a1, r,soma;
	cout<<"Primeiro termo: ";
	cin>>a1;
	cout<<"Razão: ";
	cin>>r;
	soma=5*a1+1*r //=a1+(a1+r)+(a1+2*r)+(a1+3*r)+(a1+4*r)
	//ou soma=(a1+(a1+4r))*5/2
	cout<<"Soma dos 5 primeiros termos da PA= "<<soma<<endl;
}
```

Exemplos de execuções:

Primeiro termo: 1
Razão: 3
Soma dos 5 primeiros termos da PA = 35

Primeiro termo: 10
Razão: 7
Soma dos 5 primeiros termos da PA = 120

Primeiro termo: -12
Razão: 9
Soma dos 5 primeiros termos da PA = 30

2) ** (casal) Afrânio tem um peso inicial Pa (e.g., 80 kg) e está engordando a uma média de Ta quilos por ano (e.g. 2 kg/ano). Enquanto isto, Ermengarda, sua amada esposa, tem um peso inicial Pe (e.g. 45kg) e está engordando a uma média de Te quilos por ano (e.g., 3kg/ano). Faça um programa que solicite de um usuário os

valores para Pa, Pe, Ta e Te, e calcule e imprima quantos anos serão necessários para que Ermengarda ultrapasse o peso de Afrânio. O programa também deve mostrar os pesos de cada membro do casal ternura neste momento importante de suas vidas. Considere que os valores fornecidos pelo usuário serão consistentes, tal que Ermengarda inicialmente é mais magra que Afrânio, mas que Ermengarda engorda mais rápido que seu marido.
Observe que a evolução do peso de cada um dos pombinhos é uma PA.

Quando os pesos estão iguais (iguais a P):
P=Pa+nTa=Pe+nTe, onde n=números de anos
nTa-nTe=Pe-Pa
n=(Pe-Pa)/(Ta-Te)

```
#include <iostream>
using namespace std;
int main()
{
	int Peso_a, Ta, Peso_e, Te, n;
	cout << "Entre com os valores:" << endl << "Pa: ";
	cin >> Peso_a;
	cout << "Ta: ";
	cin >> Ta;
	cout << "Pe: ";
	cin >> Peso_e;
	cout << "Te: ";
	cin >> Te;
	n=(Peso_e - Peso_a)/(Ta - Te); //n=números de anos para os pesos ficarem
IGUAIS
	n=n+1; //n=ano seguinte, quando Ermengarda ultrapassa Afranio
	cout << endl << "Ermengarda ultrapassa o peso de Afrânio em "<< n <<"
anos";
	Peso_a = Peso_a + n*Ta; //Peso de Afranio quando Ermengarda ultrapassa
	Peso_e = Peso_e + n*Te;
	cout << endl << "Peso de Afranio: "<< Peso_a << endl << "Peso de
Ermengarda: " << Peso_e << endl;
}
```

Exemplos de execuções:

```
Entre com os valores:
Pa: 80
Ta: 2
Pe: 45
Te: 3
```

```
Ermengarda ultrapassa o peso de Afrânio em 36 anos
Peso de Afranio: 152
Peso de Ermengarda: 153

Entre com os valores:
Pa: 92
Ta: 1
Pe: 54
Te: 2

Ermengarda ultrapassa o peso de Afrânio em 39 anos
Peso de Afranio: 131
Peso de Ermengarda: 132

Entre com os valores:
Pa: 55
Ta: 3
Pe: 47
Te: 4

Ermengarda ultrapassa o peso de Afrânio em 9 anos
Peso de Afranio: 82
Peso de Ermengarda: 83
```

3) * (distpontos) Construa um programa que, tendo como dados de entrada dois pontos quaisquer no plano, P(x1,y1) e P(x2,y2), escreva a distância entre eles. A fórmula que efetua tal cálculo é:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
#include <iostream>
#include <cmath>
using namespace std;


int main()
{
	float x1, y1, x2, y2, d;
	cout<<"x1: ";
	cin>>x1;
	cout<<"y1: ";
	cin>>y1;
	cout<<"x2: ";
	cin>>x2;
```

```
        cout<<"y2: ";
        cin>>y2;
        d=sqrt( pow((x2-x1),2) + pow((y2-y1),2) );
        cout<<endl<<"Distancia entre P1 e P2: "<<d<<endl;
}
```

Exemplos de execuções:

x1: -1
y1: 0
x2: 1
y2: 0

Distancia entre P1 e P2: 2

x1: 7
y1: 3
x2: 2
y2: 0

Distancia entre P1 e P2: 5.83095

x1: 7
y1: 3
x2: 2
y2: 0

Distancia entre P1 e P2: 5.83095

4) * (calcfrac) Escreva um programa que leia três números inteiros e positivos (A,B,C) e calcule a seguinte expressão:

$$D = \frac{R+S}{2}$$, onde $R = (A + B)^2$ e $S = (B + C)^2$.

```
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
        int A, B, C;
        float D, R, S;

        cout << "Indique os 3 valores inteiros pos.: ";
        cin >> A >> B >> C;
        R = pow ((A + B), 2);
```

```
        S = (B + C) * (B + C);
        D = (R + S) / 2;
        cout << "D = " << D << endl;
}
```

Exemplos de execuções:

```
Indique os 3 valores inteiros pos.: 1 2 3
D = 17

Indique os 3 valores inteiros pos.: 15 - 25 7
D = 212

Indique os 3 valores inteiros pos.: 20 37 54
D = 5765
```

5) ** (iddias) Faça um programa que leia a idade de uma pessoa expressa em anos, meses e dias e mostre-a expressa apenas em dias.

```
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
        int anos, meses, dias;
        cout << "Indique idade: ";
        cin >> anos >> meses >> dias;
        dias = anos * 365 + meses * 30 + dias;
        cout << "Sua idade em dias eh: " << dias << endl;
}
```

Exemplos de execuções:

```
Indique idade: 20 3 25
Sua idade em dias eh: 7415

Indique idade: 19 10 3
Sua idade em dias eh: 7238

Indique idade: 19 11 15
Sua idade em dias eh: 7280
```

6) * (idanos**) Faça um programa que leia a idade de uma pessoa expressa em dias e mostre-a expressa em anos, meses e dias.

```
#include <iostream>
#include <cmath>
using namespace std;

int main()

{
	int dias, anos, meses;
	cout << "Indique idades em dias: ";
	cin >> dias;
	anos = dias / 365;
	meses = (dias % 365) / 30;
	dias = (dias % 365) % 30;
	cout << "Idade = " << anos << " anos ";
	cout << meses << " meses e " << dias << " dias." << endl;

}
```

Exemplos de execuções:

```
Indique idades em dias: 7280
Idade = 19 anos 11 meses e 15 dias.

Indique idades em dias: 7777
Idade = 21 anos 3 meses e 22 dias.

Indique idades em dias: 7212
Idade = 19 anos 9 meses e 7 dias.
```

7) (mediaarit) Faça um programa que leia 3 notas de um aluno e calcule a média final do aluno. Considere que as notas têm pesos iguais.

```
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
	float a, b, c, media;
	cout << "Entre com as tres notas: ";
	cin >> a >> b >> c;
	media=(a+b+c)/3;
	cout << endl << "Media: " << media << endl;
}
```

Exemplo de execuções:

Entre com as tres notas: 3 8 10
Media: 7

Entre com as tres notas: 4.5 7 8.4
Media: 6.63333

Entre com as tres notas: 9 8.7 7.9
Media: 8.53333

8) (mediapond) Faça um programa que leia as 3 notas de um aluno e calcule a média final deste aluno. Considerar que a média é ponderada e que o peso das notas é: 2, 3 e 5, respectivamente.

```
#include <iostream>
#include <cmath>
using namespace std;

int main()

{
	float a, b, c, media;
	cout << "Entre com as tres notas: ";
	cin >> a >> b >> c;
	media=(a*2+b*3+c*5)/10;
	cout << endl << "Media: " << media << endl;

}
```

Exemplo de execuções:

Entre com as tres notas: 1 5 7
Media: 5.2

Entre com as tres notas: 8 8 7
Media: 7.5

Entre com as tres notas: 4 5 2
Media: 3.3

9) ** (convtempo) Faça um programa que leia o tempo de fabricação de um carro em uma fábrica de automóveis expressa em segundos e mostre-o expresso em horas, minutos e segundos.

```
#include <iostream>
#include <cmath>
using namespace std;

int main()
{

	int hh, mm, ss, rr, total;
	cout << "Indique tempo de fabricacao: ";
	cin >> total;
	hh = total / 3600;
	rr = total % 3600;
	mm = rr / 60;
	ss = rr % 60;
	cout << "Tempo = " << hh <<"H ";
	cout << mm << "M " << ss << "SS " << endl;

}
```

Exemplo de execuções:

```
Indique tempo de fabricacao: 13800
Tempo = 3H 50M 0SS

Indique tempo de fabricacao: 20000
Tempo = 5H 33M 20SS

Indique tempo de fabricacao: 80000
Tempo = 22H 13M 20SS
```

10) * (troco)** Escrever um programa que lê um valor em reais e calcula qual o menor número possível de notas de 100, 50, 10, 5 e 1 em que o valor lido pode ser decomposto. Escrever o valor lido e a relação de notas necessárias.

```
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
	int valor, x;
	cout << "Entre com o valor do troco em reais: ";
	cin >> valor;
	cout << endl << "Numero de notas:" << endl;
```

```
        x=valor/100;
        cout << "100: " << x << endl;
        x=valor%100/50;
        cout << "50: " << x << endl;
        x=valor%100%50/10;
        cout << "10: " << x << endl;
        x=valor%100%50%10/5;
        cout << "5: " << x << endl;
        x=valor%100%50%10%5;
        cout << "1: " << x << endl;
}
```

Exemplo de execuções:

Entre com o valor do troco em reais: 9

Numero de notas:
100: 0
50: 0
10: 0
5: 1
1: 4

Entre com o valor do troco em reais: 83

Numero de notas:
100: 0
50: 1
10: 3
5: 0
1: 3

Entre com o valor do troco em reais: 4332

Numero de notas:
100: 43
50: 0
10: 3
5: 0
1: 2

11) ** (trocadig) Dado um número inteiro não nulo de três dígitos, imprimir este número ao contrario, isto e, se a entrada for 123 (cento e vinte e três), imprimir 321 (trezentos e vinte e um). Usar operações sobre inteiros, por exemplo, divisões sucessivas por 10.

```
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
	int unidade, dezena, centena, numero;
	cout << “Indique numero: “;
	cin >> numero;
	centena = numero / 100;
	dezena = (numero % 100) / 10;
	unidade = (numero % 100) % 10;
	numero = unidade * 100 + dezena * 10 + centena;
	cout << “Numero invertido: “ << numero << endl;
}
```

Exemplos de execuções:

```
Indique numero: 653
Numero invertido: 356

Indique numero: 729
Numero invertido: 927

Indique numero: 451
Numero invertido: 154
```

12) ** (novonum) Dado um número de três dígitos, construir outro número de quatro dígitos com a seguinte regra: a) os três primeiros dígitos, contados da esquerda para a direita, são iguais aos do número dado; b) o quarto dígito é um dígito de controle calculado da seguinte forma: primeiro dígito + 3*segundo dígito + 5*terceiro dígito; o dígito de controle é igual ao resto da divisão dessa soma por 7.

```
#include <iostream>
using namespace std;

int main()
{
	int n, U, D, C;
	cout << "Entre com o numero: ";
	cin >> n;
	U=n/100;
	D=n%100/10;
```

```
        C=n%10;
        n=n*10; //O numero ganha mais um digito, e os algarismos vao para
esquerda
        n=n+(U+3*D+5*C)%7;
        cout<<"Novo numero: "<<n;
}
```

Exemplos de execuções:

Entre com o numero: 654
Novo numero: 6546

Entre com o numero: 938
Novo numero: 9382

Entre com o numero: 329
Novo numero: 3295

13) * (divint) Dados dois números inteiros positivos determinar quantas vezes o primeiro divide exatamente o segundo. Se o primeiro não divide o segundo o número de vezes é zero.

```
#include <iostream>
using namespace std;

int main()
{
        int a, b;
        cout<<”Entre com dois numeros inteiros positivos: “;
        cin>>a>>b;
        cout<<”O primeiro divide o segundo “<<b/a<<” vezes”<<endl;
}
```

Exemplos de execuções:

Entre com dois numeros inteiros positivos: 4 5
O primeiro divide o segundo 1 vezes

Entre com dois numeros inteiros positivos: 3 27
O primeiro divide o segundo 9 vezes

Entre com dois numeros inteiros positivos: 2 31264
O primeiro divide o segundo 15632 vezes

14) (divreal) Dados dois números reais positivos determinar o quociente inteiro do primeiro pelo segundo usando apenas os operadores aritméticos reais.

```
#include <iostream>
using namespace std;

int main()
{
	int quociente;
	float a, b;
	cout<<"Entre com dois numeros reais positivos: ";
	cin>>a>>b;
	quociente = b/a;	// ao atribuir um valor real à uma variável inteira o valor é “truncado”
	cout<<"O primeiro divide o segundo "<<quociente<<" vezes"<<endl;
}
```

Exemplos de execução:

```
Entre com dois numeros reais positivos: 4.3 9.838
O primeiro divide o segundo 2 vezes

Entre com dois numeros reais positivos: 7.432 98.312
O primeiro divide o segundo 13 vezes
Entre com dois numeros reais positivos: 3.982 45.632
O primeiro divide o segundo 11 vezes
```

15) (intfrac) Dado um número real positivo determinar sua parte inteira e sua parte fracionária usando apenas os operadores aritméticos reais.

16) ** (calote) João recebeu seu salário e precisa pagar duas contas atrasadas. Por causa do atraso, ele deverá pagar multa de 2% sobre cada conta. Faça um programa que calcule e mostre quanto restará do salário de João após pagar as contas. O programa deve ler o valor do salário recebido por João e do valor de cada conta que ele deveria ter pago.

```
#include <iostream>
using namespace std;

int main()
{
	float salario, conta1, conta2;
	cout<<"Entre com o salario: ";
```

```
	cin>>salario;
	cout<<"Valor da primeira conta: ";
	cin>>conta1;
	cout<<"Valor da segunda conta: ";
	cin>>conta2;
	cout<<endl<<"Restante do salario: "<<salario-conta1*1.02-
conta2*1.02<<endl;
}
```

Exemplos de execuções:

```
Entre com o salario: 500
Valor da primeira conta: 50
Valor da segunda conta: 50

Restante do salario: 398

Entre com o salario: 987
Valor da primeira conta: 52
Valor da segunda conta: 283

Restante do salario: 645.3

Entre com o salario: 1345
Valor da primeira conta: 127
Valor da segunda conta: 460

Restante do salario: 746.26
```

17) ** (ilumina) Sabe-se que, para iluminar de maneira correta os cômodos de uma casa, para cada m2 deve-se usar 18W de potência. Faça um programa que recebe as duas dimensões de um cômodo (em metros), calcule e mostre a sua área (em m2) e a potência de iluminação que deverá ser usada.

```
#include <iostream>
using namespace std;

int main()
{
	float d1, d2, area;
	cout<<"Entre com as dimensoes do comodo (em metros): ";
	cin>>d1>>d2;
	area=d1*d2;
	cout<<endl<<"Area: "<<area<<" m2"<<endl<<"Potencia: "<<area*18<<" W"
<<endl;
```

```
}
```

Exemplos de execuções:

```
Entre com as dimensoes do comodo (em metros): 4 7

Area: 28 m2
Potencia: 504 W

Entre com as dimensoes do comodo (em metros): 5 5

Area: 25 m2
Potencia: 450 W

Entre com as dimensoes do comodo (em metros): 2 3

Area: 6 m2
Potencia: 108 W
```

EXERCÍCIOS: 2 – SELEÇÃO

1) ** (eqgrau2) Fazer um programa que leia os coeficientes reais A, B e C de uma equação do 2º grau da forma

$$Ax^2 + Bx + C = 0.$$

Após a leitura o programa deve calcular e mostrar as duas raízes da equação. Caso a equação não tenha raízes reais, isto deve ser indicado na saída.

As raízes de uma equação do segundo grau são calculadas pela fórmula de Bhaskara:

$$x = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}.$$

Sabe-se também que se o valor do Discriminante B2 – 4AC for menor que 0 (zero), a equação não possui raízes reais.

```
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
	float A, B, C, delta, x1, x2;
	cout<<"Ax2 + Bx + C = 0"<<endl<<endl<<"Entre com os
coeficientes:"<<endl<<"A: ";
	cin>>A;
	cout<<"B: ";
	cin>>B;
	cout<<"C: ";
	cin>>C;
	delta = (B*B)-(4*A*C);
	if (delta>=0)
	{
		cout<<endl<<"Raizes: ";
		x1=(-B + sqrt(delta))/(2*A);
		x2=(-B - sqrt(delta))/(2*A);
		cout<<x1<<" "<<x2<<endl;
	}
	else //quando delta<0;
		cout<<"Nao existem raizes reais"<<endl;
```

```
}
```

Exemplos de execuções:

```
Ax2 + Bx + C = 0

Entre com os coeficientes:
A: 1
B: 3
C: 5
Nao existem raizes reais

Ax2 + Bx + C = 0

Entre com os coeficientes:
A: 1
B: 0
C: -4

Raizes: 2 -2

Ax2 + Bx + C = 0

Entre com os coeficientes:
A: 2
B: 12
C: 4

Raizes: -0.354249 -5.64575
```

2) ** (notas) Escreva um programa que leia o código de um aluno e suas três notas. Calcule a média ponderada do aluno, considerando que o peso para a maior nota seja 4 e para as duas restantes, 3. Mostre o código do aluno, suas três notas, a média calculada e uma mensagem "APROVADO" se a média for maior ou igual a 5 e "REPROVADO" se a média for menor que 5.

```
#include <iostream>
using namespace std;

int main()
{
	float n1, n2, n3, mp;
	int cod;

	cout << "Aluno: ";
	cin >> cod;
```

```
        cout << "Notas: ";
        cin >> n1 >> n2 >> n3;
        if (n1 >= n2 and n1 >= n3)
                mp = (n1*4 + n2*3 + n3*3) / 10;
        else if (n2 >= n1 and n2 >= n3)
                mp = (n2*4 + n1*3 + n3*3) / 10;
        else
                mp = (n3*4 + n1*3 + n2*3) / 10;

        cout <<"Codigo: "<< cod << endl << "Primeira nota: "
 << n1 << endl << "Sedunda nota: " << n2 << endl << "Terceira nota: "
 << n3 << endl;
        cout << "Media: " << mp << endl;

        if (mp >= 5)
                cout << "- Aprovado" << endl;
        else
                cout << "- Reprovado" << endl;
}
```

Exemplos de execuções:

Aluno: 12
Notas: 7 6 6
Codigo: 12
Primeira nota: 7
Sedunda nota: 6
Terceira nota: 6
Media: 6.4
- Aprovado

Aluno: 5
Notas: 3 5 8
Codigo: 5
Primeira nota: 3
Sedunda nota: 5
Terceira nota: 8
Media: 5.6
- Aprovado

Aluno: 8
Notas: 3 4 2
Codigo: 8
Primeira nota: 3
Sedunda nota: 4
Terceira nota: 2
Media: 3.1
- Reprovado

3) **** (calcnota)** Escrever um programa que lê o número de identificação, as 3 notas obtidas por um aluno em 3 provas e a média dos exercícios (*ME*) que fazem parte da avaliação. Calcular a média de aproveitamento, usando a fórmula:

$$MA = (Nota_1 + Nota_2 \times 2 + Nota_3 \times 3 + ME) / 7$$

A atribuição de conceitos obedece a tabela abaixo:

MA	Conceito
Media > 90	A
75 < media ≤ 90	B
60 < media ≤ 75	C
40 < media ≤ 60	D
Media ≤ 40	E

O programa deve escrever o número do aluno, suas notas, a média dos exercícios, a média de aproveitamento, o conceito correspondente e a mensagem: APROVADO se o conceito for A,B ou C e REPROVADO se o conceito for D ou E.

```
#include <iostream>
using namespace std;

int main()
{
	int id, a, b, c, ME, media;
	cout<<"Aluno: ";
	cin>>id;
	cout<<"Nota 1: ";
	cin>>a;
	cout<<"Nota 2: ";
	cin>>b;
	cout<<"Nota 3: ";
	cin>>c;
	cout<<"ME: ";
	cin>>ME;
	media=(a+2*b+3*c+ME)/7;
```

```
	cout<<endl<<endl<<"Aluno: "<<id<<endl<<"Notas: "<<a<<" "<<b<<"
"<<c<<endl<<"Exercicios: "<<ME<<endl<<"Media de Aproveitamento:
"<<media<<endl<<"Conceito: ";
	if (media>90)
		cout<<"A"<<endl<<"APROVADO";
	else if(media>75)
		cout<<"B"<<endl<<"APROVADO";
	else if(media>60)
		cout<<"C"<<endl<<"APROVADO";
	else if(media>40)
		cout<<"D"<<endl<<"REPROVADO";
	else
		cout<<"E"<<endl<<"REPROVADO";
	cout<<endl;
}
```

Exemplos de Execuções:

Aluno: 18
Nota 1: 75
Nota 2: 48
Nota 3: 90
ME: 83

Aluno: 18
Notas: 75 48 90
Exercicios: 83
Media de Aproveitamento: 74
Conceito: C
APROVADO

Aluno: 22
Nota 1: 65
Nota 2: 42
Nota 3: 19
ME: 50

Aluno: 22
Notas: 65 42 19
Exercicios: 50
Media de Aproveitamento: 36
Conceito: E
REPROVADO

Aluno: 11
Nota 1: 90
Nota 2: 86
Nota 3: 95

ME: 100

Aluno: 11
Notas: 90 86 95
Exercicios: 100
Media de Aproveitamento: 92
Conceito: A
APROVADO

4) ** (medias) Um usuário deseja um programa onde possa escolher que tipo de média deseja calcular a partir de 3 notas. Faça um programa que leia as notas, a opção escolhida pelo usuário e calcule as médias:

- aritmética
- ponderada (pesos 3, 3 e 4)
- harmônica (definida como sendo o número de termos dividido pela soma dos inversos de cada termo)

```
#include <iostream>
using namespace std;

int main()
{
	int t;
	float a, b, c, media;
	cout<<"Medias:"<<endl<<"1: aritmetica"<<endl<<"2: ponderada (pesos 3, 3 e
4)"<<endl<<"3: harmonica"<<endl;
	cout<<"Insira o tipo de media: ";
	cin>>t;
	cout<<"Insira as 3 notas: ";
	cin>>a>>b>>c;
	if (t==1)
		media=(a+b+c)/3;
	else if(t==2)
		media=(a*3+b*3+c*4)/(3+3+4) ;
	else if(t==3)
		media=3/(1/a+1/b+1/c);
	cout<<endl<<"media: "<<media<<endl;
}
```

Exemplos de execuções:

Medias:
1: aritmetica
2: ponderada (pesos 3, 3 e 4)

```
3: harmonica
Insira o tipo de media: 1
Insira as 3 notas: 65 65 60

media: 63.3333

Medias:
1: aritmetica
2: ponderada (pesos 3, 3 e 4)
3: harmonica
Insira o tipo de media: 2
Insira as 3 notas: 65 65 60

media: 63

Medias:
1: aritmetica
2: ponderada (pesos 3, 3 e 4)
3: harmonica
Insira o tipo de media: 3
Insira as 3 notas: 65 65 60

media: 63.2432
```

5) * (multiplos) Elaborar um programa que lê 2 valores a e b e os escreve com a mensagem: “São múltiplos” ou “Não são múltiplos”.

```
#include <iostream>
using namespace std;

int main()
{
	int a, b;
	cout<<"Entre com 2 valores: ";
	cin>>a>>b;
	if (a%b==0 || b%a==0 )
		cout<<"Os valores sao multiplos"<<endl;
	else
		cout<<"Os valores nao sao multiplos"<<endl;
}
```

Exemplos de execução:

```
Entre com 2 valores: 5 15
Os valores sao multiplos
```

Entre com 2 valores: 6 12
Os valores sao multiplos

Entre com 2 valores: 36 17
Os valores nao sao multiplos

6) **(natacao) Elabore um programa que dada a idade de um nadador classifica-o em uma das seguintes categorias:

- infantil A = 5 - 7 anos
- infantil B = 8-10 anos
- juvenil A = 11-13 anos
- juvenil B = 14-17 anos
- adulto = maiores de 18 anos

```
#include <iostream>
using namespace std;

int main()
{
	int idade;
	cout<<”Idade: “;
	cin>>idade;
	cout<<endl<<”Categoria: “;
	if (idade<=7 && idade>=5)
		cout<<”infantil A” ;
	else if(idade <=10)
		cout<<”infantil B” ;
	else if (idade <=13)
		cout<<”juvenil A” ;
	else if (idade <=17)
		cout<<”juvenil B” ;
	else
		cout<<”adulto” ;
	cout<<endl;
}
```

Exemplos de execuções:

Idade: 18
Categoria: adulto

Idade: 10

Categoria: infantil B

Idade: 14
Categoria: juvenil B

7) * (achamaior)** Escreva um programa que leia 3 números inteiros e mostre o maior deles.

```
#include <iostream>
using namespace std;

int main ()
{
   int a, b, c;

   cout << "Entre com tres numeros inteiros: ";
   cin >> a >> b >> c;

   if ( a >= b && a >= c) /* Verifica se 'a' tem o maior valor */
   {
      cout << "O maior valor eh " << a << endl;
   }

   else if ( b >= a && b >= c) /* Se 'a' não tem o maior valor, verifica
                                          se 'b' tem o maior valor */
   {
        cout << "O maior valor eh " << b << endl;
   }

   else /*  Se nem 'a' nem 'b' tem o maior valor,
               então 'c' tem  o maior valor */
   {
       cout << "O maior valor eh " << c << endl;
   }

}
```

Exemplos de execuções:

Entre com tres numeros inteiros: 1 3 2
O maior valor eh 3

Entre com tres numeros inteiros: -12 -56 -4
O maior valor eh -4

Entre com tres numeros inteiros: 0 1 -1
O maior valor eh 1

8) * (parimpar) Faça um programa que leia um número inteiro e mostre uma mensagem indicando se este número é positivo ou negativo e se ele é par ou ímpar.

```
#include <iostream>
using namespace std;

int main()
{
	int x;
	cout<<"Insira um inteiro: ";
	cin>>x;
	cout<<"O numero eh ";
	if (x%2==0)
		cout<<"par ";
	else
		cout<<"impar ";
	cout<<"e ";
	if (x>=0)
	cout<<"positivo" ;
	else
		cout<<"negativo" ;
	cout<<endl;
}
```

Exemplos de execuções:

Insira um inteiro: -44
O numero eh par e negativo

Insira um inteiro: 3
O numero eh impar e positivo

Insira um inteiro: 101
O numero eh impar e positivo

9) * (pesoideal) Tendo como dados de entrada a altura e o sexo de uma pessoa (1 masculino e 0 feminino), construa um programa que calcule seu peso ideal, utilizando as seguintes fórmulas:

- para homens: (72.7 * h) - 58
- para mulheres: (62.1 * h) - 44.7

```
#include <iostream>
#define M 0
#define H 1
using namespace std;

int main()
{
	float h;
	int s;
	cout<<"Entre com: "<<endl<<"Altura (em metros): ";
	cin>>h;
	cout<<"Sexo ("<<M<<" para feminino e "<<H<<" para masculino): " ;
	cin>>s;
	cout<<endl<<"Seu peso ideal: ";
	if (s==M)
		cout<<62.1*h-44.7;
	else
		cout<<72.7*h-58;
	cout<<endl;
}
```

Exemplos de execução:

Entre com:
Altura (em metros): 1.8
Sexo (0 para feminino e 1 para masculino): 1

Seu peso ideal: 72.86

Entre com:
Altura (em metros): 1.65
Sexo (0 para feminino e 1 para masculino): 0

Seu peso ideal: 57.765

Entre com:
Altura (em metros): 2
Sexo (0 para feminino e 1 para masculino): 1

Seu peso ideal: 87.4

10) * (divida)** Um vendedor necessita de um programa que calcule o preço total devido por um cliente, que comprou um produto em sua loja. O programa deve receber o código do produto e a quantidade comprada, para então calcular e imprimir na tela o preço total, usando a tabela abaixo:

Código do Produto	Preço unitário
1001	5,32
1324	6,45
6548	2,37
0987	5,32
7623	6,45

```
#include <iostream>
#define C1 1001
#define P1 5.32
#define C2 1324
#define P2 6.45
#define C3 6548
#define P3 2.37
#define C4 987
#define P4 5.32
#define C5 7623
#define P5 6.45
using namespace std;

int main()
{
	int c, q;
	cout<<"Codigo: ";
	cin>>c;
	cout<<"Quantidade: ";
	cin>>q;
	cout<<endl<<"Preco total= ";
	if (c==C1)
		cout<<q*P1;
	else if(c==C2)
		cout<<q*P2;
	else if(c==C3)
		cout<<q*P3;
	else if(c==C4)
		cout<<q*P4;
	else if(c==C5)
		cout<<q*P5;
	cout<<endl;
}
```

Exemplos de execuções:

Codigo: 0987
Quantidade: 3

Preco total= 15.96

Codigo: 1001
Quantidade: 1

Preco total= 5.32

Codigo: 7623
Quantidade: 9

Preco total= 58.05

11) ** (tipotriang) Elaborar um programa que lê 3 valores inteiros e positivos *a,b,c* e verifica se eles formam (sendo *a,b,c* arestas do triângulo) ou não um triângulo. Caso os valores formem um triângulo, deve-se calcular e imprimir o valor do perímetro do triângulo, indicando com uma mensagem se este é equilátero (três lados iguais), isósceles (apenas 2 lados iguais) ou escaleno (todos os lados são diferentes). Se os valores não formam um triângulo, escrever uma mensagem informando o fato. Lembre que em um triângulo, o comprimento de cada lado deve ser menor que a soma dos outros dois lados.

```
#include <iostream>
using namespace std;

int main()
{
	int a, b, c, p;
	cout<<"Insira as supostas arestas do triangulo: ";
	cin>>a>>b>>c;
	if ( ((a+b)< c) || ((b+c) < a) || ((a+c)< b) || (a==0)|| (b==0)||(c==0))
		cout<<"Nao eh triangulo."<<endl;
	else
	{
		cout<<"Eh um triangulo";
		if (a==b && b==c )
			cout<<" equilatero."<<endl;
		else if (a==b || a==c || b==c)
			cout<<" isosceles."<<endl;
		else
			cout<<" escaleno."<<endl;
	}
}
```

Exemplos de execução:

Insira as supostas arestas do triangulo: 100 1 2
Nao eh triangulo.

Insira as supostas arestas do triangulo: 4 4 5
Eh um triangulo isosceles.

Insira as supostas arestas do triangulo: 1 2 3
Eh um triangulo escaleno.

12) * (ordena)** Escrever um programa que lê um conjunto de 4 valores *i,a,b,c*, onde *i* é um valor inteiro e positivo e *a,b,c*, são quaisquer valores reais e os escreva na tela da seguinte forma:

- os três valores *a,b,c* em ordem crescente, se *i* = 1
- os três valores *a,b,c* em ordem decrescente, se *i* = 2,
- os três valores *a,b,c* de forma que o maior dentre *a,b,c* fique entre os outros dois valores, se *i* = 3.

```
#include <iostream>
using namespace std;

int main()
{
	int t;
	float a, b, c, aux;
	cout<<"Entre com os tres valores: ";
	cin>>a>>b>>c;
	cout<<"Tipo (1: crescente; 2: descrescente; 3: maior no centro): ";
	cin>>t;
	if (t==1)
	{
		if (b<a)
		{
			aux=a;
			a=b;
			b=aux;
		}
		if (c<b)
		{
			aux=c;
			c=b;
```

```
                b=aux;
        }
        if (b<a)
        {
                aux=a;
                a=b;
                b=aux;
        }
}
if (t==2)
{
        if(b>a)
        {
                aux=a;
                a=b;
                b=aux;
        }
        if(c>b)
        {
                aux=c;
                c=b;
                b=aux;
        }
        if(b>a)
        {
                aux=b;
                b=a;
                a=aux;
        }
}

if (t==3)
{
        if (a>b)
        {
                aux=a;
                a=b;
                b=aux;
        }
        if (c>b)
        {
                aux=c;
                c=b;
                b=aux;
        }
}

cout<<endl<<a<<" , "<<b<<" , "<<c<<endl;
```

```
}
```

Exemplos de execuções:

```
Entre com os tres valores: 1 2 3
Tipo (1: crescente; 2: descrescente; 3: maior no centro): 3

1 , 3 , 2

Entre com os tres valores: 3 2 1
Tipo (1: crescente; 2: descrescente; 3: maior no centro): 1

1 , 2 , 3

Entre com os tres valores: -1 -5 12
Tipo (1: crescente; 2: descrescente; 3: maior no centro): 2

12 , -1 , -5
```

13) ** (poligono) Escreva um programa para ler o número de lados de um polígono regular, e a medida do lado. Calcular e imprimir o seguinte:

- se o número de lados for 3, escrever TRIANGULO e o valor do seu perímetro;
- se o número de lados for 4, escrever QUADRADO e o valor da sua área;
- se o número de lados for 5, escrever PENTAGONO;
- se o número de lados for menor que 3 escrever a mensagem "Nao e' um poligono";
- se o número de lador for maior que 5 escrever a mensagem "Poligono nao identificado".

```
#include <iostream>
using namespace std;

int main()
{
   int n, a;
   cout<<"Numero de lados do poligono regular: ";
   cin>>n;
   cout<<endl;
   if (n==3)
   {
      cout<<"TRIANGULO"<<endl<<"Valor da aresta: ";
      cin>>a;
```

```
        cout<<"Perimetro= "<<3*a;
    }
    else if (n==4)
    {
        cout<<"QUADRADO"<<endl<<"Valor da aresta: ";
        cin>>a;
        cout<<"Area= "<<a*a;
    }
    else if (n==5)
        cout<<"PENTAGONO";
    else if (n<3)
        cout<<"Nao eh um poligono.";
    else
        cout<<"Poligono nao identificado.";
    cout<<endl;
}
```

Exeplos de execuções:

Numero de lados do poligono regular: 4

QUADRADO
Valor da aresta: 2
Area= 4

Numero de lados do poligono regular: 2

Nao eh um poligono.

Numero de lados do poligono regular: 20

Poligono nao identificado.

14) ** (somamaiores) Escreva um programa para ler 3 valores numéricos e escrever a soma dos dois maiores.

```
#include <iostream>
using namespace std;

int main ()
{
  int a, b, c, soma;
  cout << "Entre com 3 valores inteiros: ";
  cin >> a >> b >> c;
```

```
  if ( a <= b && a <= c ) // Verifica se 'a' tem o menor valor
  {
    cout << "Os maiores valores sao " << b << " e " << c << endl;
    soma = b + c;
  }
  else if ( b <= a && b <= c ) // Verifica se 'b' tem o menor valor
  {
    cout << "Os maiores valores sao " << a << " e " << c << endl;
    soma = a + c;
  }
  else //  Senão 'c' tem o menor valor
  {
    cout << "Os maiores valores sao " << a << " e " << b << endl;
    soma = a + b;
  }
  cout << "A soma deles eh " << soma << endl;
}
```

Exemplo de execução:

```
Entre com 3 valores inteiros: 1 2 3
Os maiores valores sao 2 e 3
A soma deles eh 5

Entre com 3 valores inteiros: -4 5 12
Os maiores valores sao 5 e 12
A soma deles eh 17

Entre com 3 valores inteiros: 0 9 1
Os maiores valores sao 9 e 1
A soma deles eh 10
```

EXERCÍCIOS: 3 – REPETIÇÃO

1) * (fatorial) Dado um número inteiro, calcular seu fatorial.

O fatorial de um número inteiro é definido como:

$n! = 1 * 2 * 3 * \ldots * (n - 1) * n$

$0! = 1$

```
#include <iostream>
        using namespace std;

main()
{
   double x, aux;
   cout<<"Entre com um inteiro maior que zero: ";
   cin>>x;
   cout<<x<<"! = ";
   aux=x-1;
   while (aux>0)
   {
      x=x*aux;
      aux=aux-1;
   }
   cout<<x<<endl;
}
```

Exemplos de execução:

```
Entre com um inteiro: 6
6! = 720

Entre com um inteiro maior que zero: 32
32! = 2.63131e+035

Entre com um inteiro maior que zero: 7
7! = 5040
```

2) * (soma0) Faça um programa que leia uma seqüência de numeros inteiros, sendo 0 a ordem de parada (ou seja, ao entrar com o número 0, a leitura de números se encerra), e calcule a soma dos números, imprimindo ao final o resultado da soma.

```
#include <iostream>
        using namespace std;

main()
{
   int x, soma;
   cout<<"Entre com uma sequencia de numeros inteiros, sendo 0 a ordem de
parada:"<<endl;
   soma=0;
   x=10; //numero diferente de zero para iniciar o loop
   while (x!=0)
   {
      cin>>x;
      soma=soma+x; //se o usuario entrar com zero, a soma nao se altera
   }
   cout<<endl<<"Soma total= "<<soma<<endl;
}
```

Exemplos de execuções:

Entre com uma sequencia de numeros inteiros, sendo 0 a ordem de parada:
1 3 4 5 10 0

Soma total= 23

Entre com uma sequencia de numeros inteiros, sendo 0 a ordem de parada:
-1 -9 54 2 0

Soma total= 46

Entre com uma sequencia de numeros inteiros, sendo 0 a ordem de parada:
754 100 -530 12 0

Soma total= 336

3) * (senha) Faça um programa que solicite ao usuário digitar uma senha (número inteiro) de 4 dígitos. Se a senha for incorreta, o programa repete o procedimento até o usuário entrar com a senha correta, e então imprime “Bem vindo”. Sugestão: declare via *define* a constante *SENHA* com valor 1234, por exemplo.

```
#include <iostream>
#define SENHA 1234
        using namespace std;

main()
```

```
{
	int x;
	x=SENHA-2; //Valor diferente de SENHA para entrar no loop
	while (x!=SENHA)
	{
		cout<<"Senha: ";
		cin>>x;
		if (x!=SENHA)
			cout<<"Senha incorreta!"<<endl;
	}
	cout<<endl<<endl<<"Bem vindo"<<endl;
}
```

Exemplos de execuções:

```
Senha: 0000
Senha incorreta!
Senha: 1111
Senha incorreta!
Senha: 1234

Bem vindo
```

4) ** (somaneg) Faça um programa que leia uma seqüência de numeros inteiros, sendo um número negativo a ordem de parada (ou seja, ao entrar com um número negativo, a leitura de números se encerra), e calcule a soma dos números lidos, imprimindo ao final o resultado da soma. O número negativo não deve ser considerado no calculo da soma.

```
#include <iostream>
	using namespace std;

main()
{
	int x, soma;
	cout<<"Entre com uma sequencia de numeros inteiros, sendo um numero
negativo a ordem de parada:"<<endl;
	soma=0;
	cin >> x;
	while (x>=0)
	{
		soma=soma+x;
		cin>>x;
	}
```

```
        cout<<endl<<"Soma= "<< soma <<endl;
}
```

Exemplos de execuções:

Entre com uma sequencia de numeros inteiros, sendo um numero negativo a ordem de
parada:
1 2 3 0 8 -7
Soma= 14

Entre com uma sequencia de numeros inteiros, sendo um numero negativo a ordem de
parada:
93 12 1 1 -4
Soma= 107

Entre com uma sequencia de numeros inteiros, sendo um numero negativo a ordem de
parada:
9 0 0 2 -1
Soma= 11

5) ** (achamaior) Fazer um programa em C++ que seja capaz de ler números inteiros positivos do teclado e de identificar o maior destes números lidos. Ao final, o maior número encontrado deve ser impresso. O número zero é o ultimo número fornecido como dado de entrada. OBSERVACAO: o zero não deve ser considerado na busca do maior valor.

```
#include <iostream>
        using namespace std;

main()
{
        int x, maior;
        cout<<”Entre com uma sequencia de inteiros positivos, sendo zero a ordem de parada”<<endl;
        maior=0; //O zero eh menor que o menor valor permitido da sequencia
        x=18; //valor diferente de zero para o programa entrar no loop
        while (x!=0)
        {
                cin>>x;
                if (x!=0 && x>0) //Zero nao deve ser considerado na busca do maior valor
```

```
            //Verifica se x eh positivo
        {
                if (x>maior)
                        maior=x;
        }
        if (x<0)
                cout<<”Numero invalido”<<endl;
    }
    cout<<endl<<endl<<”Maior: “<<maior<<endl;
}
```

Exemplos de execuções:

Entre com uma sequencia de inteiros positivos, sendo zero a ordem de parada
1 2 3 4 0
Maior: 4

Entre com uma sequencia de inteiros positivos, sendo zero a ordem de parada
12 32 6 9 0
Maior: 32

Entre com uma sequencia de inteiros positivos, sendo zero a ordem de parada
78 1 2 65 0
Maior: 78

6) ** (serie) Faça um programa em C++ que lê um número inteiro *N*, calcule e imprima a soma dos termos da série abaixo, para *N* termos:

$$S=\frac{5}{2}+\frac{9}{4}+\frac{17}{8}+\frac{33}{16}+\frac{65}{32}+\ldots$$

```
#include <iostream>
    using namespace std;

main()
{
    float S, numerador, denominador;
    int N, i;
    cout<<"Entre com o inteiro N: ";
    cin>>N;
    S=0;
    numerador=5;
    denominador=2;
```

```
        i=N;
        while (i>0)
        {
                S=S+(numerador/denominador);
                numerador=numerador+(numerador - 1);
                denominador=denominador*2;
                i=i-1;
        }
        cout<<"S= "<<S<<endl;
}
```

Exemplos de execução:

```
Entre com o inteiro N: 4
S= 8.9375

Entre com o inteiro N: 12
S= 24.9998

Entre com o inteiro N: 27
S= 55
```

7) * (coseno1)** Considere a soma *S* das parcelas apresentadas abaixo, as quais são originadas de uma série infinita:

$$S = 1 - \frac{1}{2!} + \frac{1}{4!} - \frac{1}{6!} + \frac{1}{8!} - \frac{1}{10!} + \frac{1}{12!} - \cdots$$

Esta soma corresponde ao cálculo do valor do co-seno de 1 (um) radiano. Fazer um programa em C++ que calcule o valor de *S* considerando apenas as 14 primeiras parcelas. Ao final, imprimir o resultado encontrado para *S*.

```
#include <iostream>
using namespace std;

main()
{
        float S;
        int i, fat, ifat;
        S=1;
        i=2;
        while (i<=13) //A primeira parcela (1) ja foi considerada
        {
                fat=(i-1)*2;
                ifat=fat-1;
```

```
            while (ifat>0)
            {
                  fat=fat*ifat;
                  ifat=ifat-1;
            }
            if (i%2==0)
                  S=S-1.0/fat; //Use 1.0 ou declare float fat
            else
                  S=S+1.0/fat; //Use 1.0 ou declare float fat
            i=i+1;
      }
      cout<<"cos (1 radiano) = "<<S<<endl;
}
```

Resultado

cos (1 radiano) = 0.540302

8) * (ehprimo)** Dado um número inteiro maior que 1, verifique se este número é primo ou não, exibindo mensagens convenientes em cada caso.

```
#include <iostream>
      using namespace std;

main()
{
      int x, aux, i;
      x=0; //Inteiro menor que 1 para entrar no loop
      while (x<=1) //checa se x eh um inteiro maior que 1
      {
            cout<<"Entre com um inteiro maior que 1: ";
            cin>>x;
      if (x<=1)
            cout<<"Numero invalido"<<endl;
      }
      i=0;
      aux=2;
      while (aux<x)
      {
            if (x%aux==0)
                  i=i+1;
            aux=aux+1;
      }
      if (i==0)
            cout<<endl<<x<<" eh primo."<<endl;
      else
```

```
            cout<<endl<<x<<” nao eh primo.”<<endl;
}
```

Exemplos de execuções:

Entre com um inteiro maior que 1: 15
15 nao eh primo.

Entre com um inteiro maior que 1: 77
77 nao eh primo.

Entre com um inteiro maior que 1: 113
113 eh primo.

9) * (fibonacci)** Faça um programa que exiba na tela os 20 primeiros números da sequência de Fibonnacci.

A sequência de Fibonnacci é definida como:
fib(0) = 0
fib(1) = 1
fib(*n* - 1) + *fib*(*n* - 2)
ou seja, a sequência 0,1,1,2,3,5,8,13,21,34

```
#include <iostream>
        using namespace std;

main()
{
        int fibn, fibn1, fibn2, n;
        cout<<"20 primeiros numeros da sequencia de Fibonnaci:"<<endl;
        n=0;
        while (n<30)
        {
                if (n==0)
        {
                fibn=0;
                fibn2=0;
                cout<<fibn;
        }
        else if (n==1)
        {
                fibn=1;
                fibn1=1;
                cout<<fibn;
        }
        else if (n>1)
```

```
        {
                fibn=fibn1+fibn2;
                cout<<fibn;
                fibn2=fibn1;
                fibn1=fibn;
        }
        cout<<",";
        n=n+1;
        }
        cout<<endl;
}
```

Resultado:

```
20 primeiros numeros da sequencia de Fibonnaci:
0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,2584,4181,6765,10946,177
11
,28657,46368,75025,121393,196418,317811,514229,
```

10) * (palindromo)** Fazer um programa em C++ que seja capaz de ler um número inteiro positivo do teclado e de verificar se a sua sequência de dígitos é exatamente a mesma, tanto se for analisado da esquerda para a direita quanto da direita para a esquerda. Caso seja, imprimir “ESSE NUMERO E PALÍNDROMO!!!”. Caso não seja, imprimir “ESSE NUMERO NÃO É PALÍNDROMO.” OBSERVAÇÃO: utilize divisões e/ou multiplicações sucessivas por 10.

```
#include <iostream>
using namespace std;

main()
{
        int x, tam, comparacao, ip, alg1, alg2, i, aux;
        cout<<"Entre com um inteiro positivo: ";
        cin>>x;
        while (x<10) //enquanto x tem apenas 1 digito
        {
                cout<<"Numero invalido. Digite um inteiro positivo COM MAIS DE UM
DIGITO: ";
        cin>>x;
        }
        tam=1;
        aux=x;
        while (aux>=10)
        {
```

```
            aux=aux/10;
        tam=tam+1;
        }
        ip=0;
        comparacao=1;
        while (comparacao<=(tam/2))
        {
            alg1=x;
            i=1;
            while (i<comparacao)
            {
                alg1=alg1/10;
                i=i+1;
            }
                alg1=alg1%10;
            alg2=x;
             i=1;
            while (i<(tam-(comparacao-1)))
            {
                alg2=alg2/10;
                i=i+1;
            }
            alg2=alg2%10;
            if (alg1==alg2)
                ip=ip+1;
            comparacao=comparacao+1;
        }
        cout<<endl<<x;
        if (ip==(tam/2))
            cout<<" eh palindromo"<<endl;
        else
            cout<<" nao eh palindromo"<<endl;
}
```

Exemplos de execuções:

Entre com um inteiro positivo: 12321
12321 eh palindromo

Entre com um inteiro positivo: 3456765
3456765 nao eh palindromo

Entre com um inteiro positivo: 12121212
12121212 nao eh palindromo

EXERCÍCIOS: 4 – FUNÇÕES COM RETORNO POR VALOR

1) ** (degraus) Cada degrau de uma escada tem X cm de altura. Faça um programa que receba esta altura em centímetros e a altura em metros que o usuário deseja alcançar subindo a escada, calcule e mostre quantos degraus ele deverá subir para atingir seu objetivo, sem se preocupar com a altura do usuário.

```
#include <iostream>
using namespace std;

int ndegraus (float Hdegrau, float Halcance)
{
	int i;
	float Hpercorrido;
	Hpercorrido=Hdegrau/100.0;
	i=1;
	while (Hpercorrido<Halcance)
	{
		Hpercorrido=Hpercorrido+Hdegrau/100.0;
		i=i+1;
	}
	return i;
}

int main()
{
	float degrau, alcance;
	int n;
	cout<<"Altura do degrau (cm): ";
	cin>>degrau;
	cout<<"Altura que deseja alcancar (m): ";
	cin>>alcance;
	n=ndegraus (degrau, alcance);
	cout<<"Para alcancar "<<alcance<<" m, subira "<<n<<" degraus."<<endl;
}
```

Exemplos de execuções:

```
Altura do degrau (cm): 10
Altura que deseja alcancar (m): 1
Para alcancar 1 m, subira 10 degraus.

Altura do degrau (cm): 15
Altura que deseja alcancar (m): 1.08
```

Para alcancar 1.08 m, subira 8 degraus.

Altura do degrau (cm): 9.5
Altura que deseja alcancar (m): 20
Para alcancar 20 m, subira 211 degraus.

2) ** (escada) Faça um programa que receba a medida do ângulo, em graus, formado por uma escada apoiada no chão e a distância em que a escada está de uma parede, calcule e mostre a altura, em relação ao chão, do ponto em que a escada toca a parede. Para calcular a altura, use uma função, que tenha como parâmetros a medida do ângulo e a distância que a escada está da parede e retorne o valor real da altura. OBS: as funções trigonométricas da biblioteca cmath calculam ângulos em radianos.

```
int main()
{
	float degrau, alcance;
	int n;
	cout<<"Altura do degrau (cm): ";
	cin>>degrau;
	cout<<"Altura que deseja alcancar (m): ";
	cin>>alcance;
	n=ndegraus (degrau, alcance);
	cout<<endl<<"Para alcancar "<<alcance<<" m, subira "<<n<<"
degraus."<<endl;
}
```

```
#include <iostream>
#include <cmath>
#define PI 3.14159265359
using namespace std;

float altura (float alfa, float d)
{
	alfa=(alfa*PI)/180; //converte graus para radianos
	return d*sin(alfa);
}

int main()
{
	float alfa, distancia, h;
	cout<<"Angulo (graus): ";
	cin>>alfa;
```

```
	cout<<"Distancia: ";
	cin>>distancia;
	h= altura (alfa, distancia);
	cout<<endl<<"Altura= "<<h<<endl;
}
```

Exemplos de execuções:

Angulo (graus): 45
Distancia: 7

Altura= 4.94975

Angulo (graus): 30
Distancia: 9

Altura= 4.5

Angulo (graus): 60
Distancia: 2

Altura= 1.73205

3) * (calculos) Faça um programa que receba do usuário um número positivo e diferente de zero, calcule e mostre:

1. O quadrado do número;
2. A raiz cúbica do número;
3. A raiz quadrada do número;
4. O cubo do número.

4) ** (potencia) Faça um programa que receba do usuário um número real positivo e diferente de zero e que calcule o quadrado do número real. Para isso, use uma função *quadrado()* que receba como parâmetro o valor real e retorne o quadrado desse número.

```
#include <iostream>
using namespace std;

float quadrado (float x)
{
	return x*x;
}
```

```
int main()
{
	float x;
	cout<<"Entre com um numero real positivo e diferente de zero: ";
	cin>>x;
	cout<<x<<" ao quadrado = "<<quadrado(x)<<endl;
}
```

Exemplos de execuções:

```
Entre com um numero real positivo e diferente de zero: 3
3 ao quadrado = 9

Entre com um numero real positivo e diferente de zero: 9
9 ao quadrado = 81

Entre com um numero real positivo e diferente de zero: 4
4 ao quadrado = 16
```

5) * (volesfera) Faça um programa que calcula e mostre o volume de uma esfera cujo diâmetro em metros é informado pelo usuário. Para isso, crie uma função *volesfera()* que receba como parâmetro o raio da esfera em metros e retorne o valor do volume da esfera. Lembre-se que o volume de uma esfera é fornecido pela fórmula $V = \frac{4\pi}{3} \times R^3$

```
#include <iostream>
#define PI 3.141592654
using namespace std;

float volesfera(float x)
{
	return ((4*PI*x*x*x)/3);
}

int main()
{
	float x=0;
	cout<<"Digite o tamanho do raio da esfera: ";
	cin>>x;
	cout<<x<<"O volume da esfera eh: = "<< volesfera(x)<<" metros cubicos." <<
endl;
}
```

Exemplos de execução:
Digite o tamanho do raio da esfera: 2
O volume da esfera eh: = 33.5103 metros cubicos.

Digite o tamanho do raio da esfera: 1
O volume da esfera eh: = 4.18879 metros cubicos.

Digite o tamanho do raio da esfera: 7.523
O volume da esfera eh: = 1783.45 metros cubicos.

6) ** (invertedig) Fazer uma função chamada troca() que receba como parâmetro dois números inteiros não nulos de 4 dígitos e retorne TRUE (1) se um for o contrário do outro e FALSE (0) em caso contrário. Isto é, se os parâmetros forem 4123 (quatro mil e cento e vinte e três) e 3214 (três mil e duzentos e quatorze), deve-se retornar TRUE (1). Usar apenas operações sobre inteiros. Faça também um programa que peça ao usuário dois valores inteiros de 4 dígitos e, usando a função troca(), mostre mensagens dizendo se o primeiro número informado pelo usuário é o contrário do segundo número.

```
#include <iostream>
using namespace std;

int troca(int a, int b)
{
	b=b/1000+b/100%10*10+b/10%10*100+b%10*1000;
//Unid+Dezen+Cent+UMilhar
	if (a==b)
		return 1;
	else
		return 0;
}

int main()
{
	int a, b, digito;
	cout<<"Entre com dois inteiros de 4 digitos: ";
	cin>>a>>b;
	digito=troca(a, b);
	if (digito==1)
		cout<<"Um eh contrario do outro.";
	else if (digito==0)
		cout<<"Um nao eh contrario do outro.";
```

```
        cout<<endl;
}
```

Exemplos de execução:

Entre com dois inteiros de 4 digitos: 5789 9857
Um nao eh contrario do outro.

Entre com dois inteiros de 4 digitos: 1234 4321
Um eh contrario do outro.

Entre com dois inteiros de 4 digitos: 1000 1000
Um nao eh contrario do outro.

7) *(bindecimal) Fazer uma função chamada convbindec() que receba como parâmetro um número inteiro representando um número binário e retorne seu valor equivalente em decimal. Por exemplo, se a entrada for 10001, a saída deve ser 17. Faça também um programa que peça ao usuário dois valores representando um número binário e, usando a função convbindec(), mostre na tela os valores decimais correspondentes a cada um dos números.

```
#include <iostream>
using namespace std;

int convbindec(int bin)
{
        int dec=0, d=1;
        while(bin != 0)
        {
                dec = dec +(bin%10)*d;
                d = d*2;
                bin = bin/10;
        }
        return dec;
}

int main()
{
        int bin1,bin2,dec1,dec2;
        cout<<"Entre com dois numeros binarios: ";
        cin >> bin1 >> bin2;
        dec1=convbindec(bin1);
        dec2=convbindec(bin2);
```

```
	cout<<"O numero "<< bin1 << " em binario é o numero "<< dec1 << " em decimal." << endl;
	cout<<"O numero "<< bin2 << " em binario é o numero "<< dec2 << " em decimal." << endl;

}
```

Exemplos de execução:

```
Entre com dois numeros binarios: 111001 101010
O numero 111001 em binario é o numero 57 em decimal.
O numero 101010 em binario é o numero 42 em decimal.

Entre com dois numeros binarios: 10 11
O numero 10 em binario é o numero 2 em decimal.
O numero 11 em binario é o numero 3 em decimal.

Entre com dois numeros binarios: 1010111 1000011
O numero 1010111 em binario é o numero 87 em decimal.
O numero 1000011 em binario é o numero 67 em decimal.
```

8) **(menor3) Fazer uma função chamada menor() que receba como parâmetro três números inteiros não nulos e retorne o menor valor dentre os três. Se os três valores foram iguais, retorna qualquer um dos valores. Faça também um programa que receba do usuário 8 números inteiros e, usando esta função, determine e mostre na tela o menor valor dentre os 8 números.

```
#include <iostream>
using namespace std;

int menor (int x, int y, int z)
{
	if(x<=y && x<z)
		return x;
	else if(y<x && y<=z)
		return y;
	else if (z<=x && z<y)
		return z;
}

int main()
{
	int a, b, c, i=0, m;    // m  guarda o menor valor da iteração
	cout << "Digite 8 numeros inteiros nao nulos: ";
```

```
        cin >> a >> b >> m;
        while(i < 2)
        {
                m = menor(m, a, b);
                cin >> a >> b;
                i = i + 1;
        }
        cin >> a;
        m = menor(m, a, b);
        cout << "O menor dentre os numeros eh: " << m << endl;
}
```

Exemplos de execuções:
Digite 8 numeros inteiros nao nulos: 2 3 5 8 7 6 5 4
O menor dentre os numeros eh: 2
Digite 8 numeros inteiros nao nulos: 9 9 8 9 8 7 89 7
O menor dentre os numeros eh: 7
Digite 8 numeros inteiros nao nulos: -1 2 3 -4 5 6 -7 8
O menor dentre os numeros eh: -7

9) ** (maior3) Fazer uma função chamada maior() que receba como parâmetro três números inteiros não nulos e retorne o maior valor dentre os três. Se os três valores foram iguais, retorna qualquer um dos valores. Faça também um programa que receba do usuário 10 números inteiros e, usando esta função, determine e mostre na tela o maior valor dentre os 10 números.

10) ** (pesoideal) Fazer uma função chamada pesoIdeal() que recebe como parâmetros a altura e o sexo de uma pessoa (**0** masculino e **1** feminino) e retorna o seu “peso” ideal, utilizando as seguintes fórmulas:

- para homens: (72.7 * *h*) - 58
- para mulheres: (62.1 * *h*) - 44.7

Faça também um programa que receba do usuário a sua altura (em metros), “peso” atual (em kg) e sexo, e imprima na tela se o usuário está acima, ou se está abaixo, ou se está igual ao seu “peso” ideal, este calculado pela função pesoIdeal().

11) ** (bolsa) Deseja-se distribuir bolsas de estudo para alunos de um curso. O valor da bolsa depende do valor da nota que o aluno teve em uma certa disciplina, conforme tabela abaixo:

Nota em disciplina	Bolsa (R$)
70 ≤ nota < 80	500,00
80 ≤ nota < 90	600,00
Nota ≥ 90	700,00
70 > nota	0,00 (sem bolsa)

Pede-se um programa que, ao receber a nota de 3 (três) alunos, escolha a maior nota e, conforme o valor desta, imprima na tela o valor da bolsa correspondente. Use a função maior() desenvolvida no exercício 9.

12) * (prnprimos)** Fazer uma função chamada ehPrimo() que receba como parâmetro um número inteiro e retorne TRUE (1) se ele for primo e FALSE (0), caso contrário. Faça também um programa que, usando esta função, imprima na tela todos os números primos entre 0 e 1000.

EXERCÍCIOS: 5 – FUNÇÕES COM PARÂMETROS DE SAÍDA

1) * (troca) Fazer uma função chamada troca() que receba dois parâmetros reais X e Y e retorne por referência as variáveis com os conteúdos trocados (usando passagem por referência. Isto é, se X=5 e Y=8, após a função, X=8 e Y=5.

```
#include <iostream>
using namespace std;

void troca(float& x, float& y)
{
	float aux;
	aux = x;
	x = y;
	y = aux;
}

int main()
{
	float x, y;
	cout << "Digite dois numeros reais:" << endl;
	cout << "x = ";
	cin >> x;
	cout << "y = ";
	cin >> y;
	troca(x,y);
	cout << "Os valores trocados de x e y sao: x= "<< x <<" e y= "<< y << endl;
}
```

Exemplos de execução:

```
Digite dois numeros reais:
x = 12
y = 44
Os valores trocados de x e y sao: x= 44 e y= 12

Digite dois numeros reais:
x = -54
y = 0
Os valores trocados de x e y sao: x= 0 e y= -54

Digite dois numeros reais:
x = 2
y = 1
Os valores trocados de x e y sao: x= 1 e y= 2
```

2) **(ordena) Faça uma função chamada ordena() que recebe 3 números inteiros *a*, *b* e *c*, e ordena os valores de forma que A passe a ter o menor valor e C o maior valor.

```
#include <iostream>
using namespace std;

void troca(int& x, int& y)
{
	int aux;
	aux = x;
	x = y;
	y = aux;
}

void ordena(int& a, int& b, int& c)
{
	if(b<a && b<c)
		troca(a, b);
	else if(c<a && c<=b)
		troca (a, c);

	if(b>c)
	troca(b,c);
}

int main()
{
	int a, b, c;
	cout << "Digite tres numeros inteiros:" << endl;
	cin >> a >> b >> c;
	ordena(a, b, c);
	cout << "Os valores ordenados sao:  "<< a << " " << b <<
	" " << c << endl;
}
```

Exemplos de execuções:

```
Digite tres numeros inteiros:
8 34 1
Os valores ordenados sao:  1 8 34

Digite tres numeros inteiros:
-1 -2 3
Os valores ordenados sao:  -2 -1 3
```

```
Digite tres numeros inteiros:
1 2 1
Os valores ordenados sao:  1 1 2
```

3) *(atribquad) Faça uma função chamada atribquad() que recebe como parâmetro um numero real representando a medida do lado de um quadrado, devolvendo 3 valores: a área, o perímetro e o tamanho da diagonal (dada por *lado* × (raiz de 2)).

```
#include <iostream>
#include <cmath>
using namespace std;
void atribquad(float lado, float& area, float& perimetro, float& diag)
{
	area = lado*lado;
	perimetro = lado*4;
	diag = lado*sqrt(2);
}
int main()
{
	float lado, area, perimetro, diag;
	cout << "Digite o valor do lado do quadrado: ";
	cin >> lado;
	atribquad( lado, area, perimetro, diag);
	cout << "Os valores sao:  " << endl;
	cout << "Area : "<< area << endl;
	cout << "Perimetro: " << perimetro << endl;
	cout << "Diagonal: " << diag << endl;
}
```

Exemplos de execuções:

```
Digite o valor do lado do quadrado: 4
Os valores sao:
Area : 16
Perimetro: 16
Diagonal: 5.65685

Digite o valor do lado do quadrado: 2.21
Os valores sao:
Area : 4.8841
Perimetro: 8.84
Diagonal: 3.12541
```

```
Digite o valor do lado do quadrado: 9
Os valores sao:
Area : 81
Perimetro: 36
Diagonal: 12.7279
```

4) ** (tempoJogo) Escreva função chamada tempoJogo() que recebe 4 parâmetros inteiros, cada par de inteiros representando dois horários (horas e minutos). A função deve devolver o período em minutos decorrido entre os dois horários, e também o valor 1 (um) se o intervalo de tempo calculado for maior que 3 horas e meia ou 0 (zero) caso contrário.

```
#include <iostream>
#include <cmath>
using namespace std;
int tempoJogo(int& horas1, int& minutos1, int horas2, int minutos2)
{
	horas1 = abs(horas2 - horas1 + );
	minutos1 = abs(minutos2 - minutos1) ;

	if ((horas1 > 3)&&(minutos1 > 30))
		return 1;
	else
		return 0;
}

int main()
{
	int h1, m1, h2, m2;
	cout << "Digite os horarios: ";
	cin >> h1 >> m1 >> h2 >> m2;

	if(tempoJogo(h1, m1, h2, m2))
		cout<< "Jogo longo. Durou exatamente "<< h1 << "h "
		<< m1 << "m. ";
	else
		cout<< "Jogo curto. Durou exatamente "<< h1 << "h "
		<< m1 << "m. ";

}
```

Exemplos de execuções:

```
Digite os horarios: 2 30
3 15
```

Jogo curto. Durou exatamente 1h 15m.

Digite os horarios: 5 00
12 41
Jogo longo. Durou exatamente 7h 41m.

Digite os horarios: 10 10
11 11
Jogo curto. Durou exatamente 1h 1m.

5) ** (maxPot) Faça uma função chamada maxpot() que recebe como parâmetros três inteiros: *X*, *Y* e *MAX*. Esta função deve devolver 2 valores: o resultado de X^Y e o valor 1 (um) se este valor calculado for maior que *MAX*, ou 0 (zero) caso contrário.

EXERCÍCIOS: 6 – VETORES

1) * (censo)** Foi realizada uma pesquisa entre N habitantes de uma certa região. De cada habitante foram coletados os dados: Idade, sexo, salário e número de filhos.

Faça um programa em linguagem C que receba do usuário uma lista de dados *(idade, sexo* (1: feminino; 0: masculino)*,salario, num_filhos)* e calcule a média salarial, a maior e menor idade do grupo, a média de idade e a quantidade de mulheres com 3 filhos que recebe até R$ 500,00. Quando a *idade* fornecida for 0 (zero), o programa deve exibir os resultados e terminar.
Use uma função para cada cálculo.

```
#include <iostream>
#define N 100
#define H 0
#define M 1
using namespace std;

int media (int v[], int limite)
{
	int soma, i;
	soma=0;
	i=0;
	while (i<=limite)
	{
		soma=soma+v[i];
		i=i+1;
	}
	return soma/(i+1);
}

int maior (int v[], int limite)
{
	int maior, i;
	maior=0;
	i=0;
	while (i<=limite)
	{
		if (v[i]>maior)
		maior=v[i];
		i=i+1;
	}
```

```
	return maior;
}

int menor (int v[], int limite)
{
	int menor, i;
	menor=10000;
	i=0;
	while (i<=limite)
	{
		if (v[i]<menor)
		menor=v[i];
		i=i+1;
	}
	return menor;
}

int nmulheres (int sexo[N], int num_filhos[N], int salario[N], int limite)
{
	int i, n;
	n=0;
	i=0;
	while (i<=limite)
	{
		if (sexo[i]==M)
	{
		if (num_filhos[i]==3)
		{
			if (salario[i]<=500)
			n=n+1;
		}
	}
	i=i+1;
	}
	return n;
}

int main()
{
	int idade[N], sexo[N], salario[N], num_filhos[N], i;

	i=0;

	cout<<"Cidadão 1:"<<endl<<"Idade: ";
	cin>>idade [i];
	while (idade[i]!=0)
	{
		cout<<"Sexo: (H-0, M-1) ";
```

```
            cin>>sexo[i];
            cout<<"Salario: RS ";
            cin>>salario[i];
            cout<<"Numero de filhos: ";
            cin>>num_filhos [i];
            i=i+1;
            cout<<"Cidadão "<< i+1 <<":"<<endl<<"Idade: ";
            cin>>idade [i];
        }

        i=i-1;
        cout<<endl<<endl<<"Media salarial= "<<media (salario, i)<<endl<<"Maior
idade= "<<maior (idade, i)<<endl<<"Menor idade= "<<menor (idade,
i)<<endl<<"Media da idade= "<<media (idade, i)<<endl<<"Num de mulheres com 3
filhos que recebe ate RS 500,00= "<<nmulheres (sexo, num_filhos, salario, i)<<endl;
}
```

Exemplo de execução:

```
Cidadão 1:
Idade: 18
Sexo: (H-0, M-1) 1
Salario: RS 400
Numero de filhos: 3
Cidadão 2:
Idade: 34
Sexo: (H-0, M-1) 0
Salario: RS 1000
Numero de filhos: 1
Cidadão 3:
Idade: 22
Sexo: (H-0, M-1) 1
Salario: RS 450
Numero de filhos: 3
Cidadão 4:
Idade: 0


Media salarial= 462
Maior idade= 34
Menor idade= 18
Media da idade= 18
Num de mulheres com 3 filhos que recebe ate RS 500,00= 2
```

2) * (somavetores) Fazer um programa que leia do teclado duas sequências de 20 inteiros em dois vetores distintos e gere um 3º-vetor em que cada elemento é igual à

soma dos elementos correspondentes dos dois vetores lidos anteriormente, imprimindo este vetor na tela após a operação. A operação de soma dos vetores deve ser feita em uma função à parte denominada soma_vetor() que recebe 4 parâmetros: 3 vetores de inteiros e um inteiro indicando o tamanho dos vetores. Esta função deve somar os dois vetores indicados nos dois primeiros parâmetros e armazenar o resultado no vetor indicado como 3º parâmetro.

```
#include <iostream>
#define N 20
using namespace std;

void levetor (int V[], int tamanho)
{
	int i;
	i=0;
	while (i<tamanho)
	{
		cin>>V[i];
		i=i+1;
	}
}

void soma_vetor (int A[], int B[], int C[], int tamanho)
{
	int i;
	i=0;
	while (i<tamanho)
	{
		C[i]=A[i]+B[i];
		i=i+1;
	}
}

void imprime_vetor (int V[], int tamanho)
{
	int i;
	i=0;
	while (i<tamanho)
	{
		cout<<V[i]<<" ";
		i=i+1;
	}
	cout << endl;
}
```

```
int main()
{
	int A[N], B[N], C[N], t;
	t=N;
	cout<<"Entre com o Vetor 1:"<<endl;
	levetor(A,N);
	cout<<"Entre com o Vetor 2:"<<endl;
	levetor(B,N);
	soma_vetor (A, B, C, t);
	cout<<endl<<endl<<"Vetor soma dos vetores 1 e 2:"<<endl;
	imprime_vetor (C,N);
}
```

Exemplo de execução:

```
Entre com o Vetor 1:
45 74 85 34 12 5 8 95 74 5 12 35 42 54 85 16 32 57 81 77
Entre com o Vetor 2:
2 3 58 54 65 12 36 54 74 55 69 99 12 42 35 14 74 18 19 25


Vetor soma dos vetores 1 e 2:
47 77 143 88 77 17 44 149 148 60 81 134 54 96 120 30 106 75 100 102
```

3) ** (lerespecialvetor) Escreva a função **obtem_vetor()** que ao ser chamada preencha um vetor de inteiros com dados do usuário. Esta função deve ter como argumentos o vetor a ser preenchido, o tamanho máximo do vetor e o valor inteiro que o usuário deve digitar indicando final da entrada de dados (este valor NÃO deve ser armazenado no vetor). A função deve retornar a quantidade de elementos lidos.

```
#include <iostream>
using namespace std;
#define N 100

int obtem_vetor (int V[], int tamanho, int stop)
{
	int i, aux;
	i=0;
	while (i<tamanho && aux!=stop)
```

```
        {
                cin>>aux;
                if (aux!=stop)
                        V[i]=aux;
                i=i+1;
        }
        return i-1;
}
int main()
{
        int V[N];
        int qtd;
        cout<<"Entre com um numero ou digite \'0\' para terminar:" <<endl;
        qtd=obtem_vetor(V,N,0);
        cout<<"A qtd de elementos lidos eh: "<< qtd<<endl;
}
```

Exemplo de execução:

```
Entre com um numero ou digite '0' para terminar:
45
14
32
54
75
65
22
0
A qtd de elementos lidos eh: 7
```

4) ** (numvezes) Escreva a função **conta_ocorr()** que recebe 2 argumentos: o 1º argumento é um vetor de caracteres e o 2º argumento é um caracter. A função deve retornar como resultado o número de vezes o 2º argumento da função ocorre dentro

do vetor. O vetor de caracteres recebido pela função possui como último elemento o valor inteiro 0 (zero), que indica o final do vetor.

```
#include <iostream>
using namespace std;

#define N 100


int conta_ocorr(char palavra[], char letra)
{
	int n, i;
	n=0;
	i=0;
	while (palavra[i]!='0')
	{
		if (letra==palavra[i])
			n=n+1;
		i=i+1;
	}
	return n;
}

int main()
{
	char V[N],C;
	int ocorrencias,i=0;
	cout<<"Entre com um vetor de caracteres com \'0\' para indicar o final:"
<<endl;
	cin>>V[i];
	while(V[i]!='0')
	{
		i=i+1;
		cin>>V[i];
	}
	cout<<"Entre com um caracter: ";
	cin>>C;
	ocorrencias=conta_ocorr(V,C);
	cout<<"A qtd de caracteres \'"<<C<<"\' encontrados eh: "<< ocorrencias
<<endl;
}
```

Exemplo de execução:

```
Entre com um vetor de caracteres com '0' para indicar o final:
x a f h t h u h h 0
Entre com um caracter: h
A qtd de caracteres 'h' encontrados eh: 4
```

5) * (soma30) Faça uma função que some todos os valores reais de um vetor com 30 elementos e retorne o valor da soma.

```
#include <iostream>
using namespace std;

#define N 30


float soma30 (float V[])
{
	int i;
	float soma;
	soma=0;
	i=0;
	while (i<30)
	{
		soma=soma+V[i];
		i=i+1;
	}
	return soma;
}

int main()
{
	float V[N],soma;
	int i=0;
	cout<<"Entre com um vetor de 30 elementos:" <<endl;
	while(i < N)
	{
		cin>>V[i];
		i=i+1;
	}
	soma=soma30(V);
	cout<<"A soma do vetor eh \'"<<soma<<"\'" <<endl;
}
```

Exemplo de execução:
Entre com um vetor de 30 elementos:
45.4 41.2 365.4 425.47 41.58 41.78 74.7 796.8 54 23.5 41.8 54.4 133.5 47.2 41.54 85.3 123.45 74.5 85.6 32.5 358.9 985.7 41 52.4 75.6 3.3 214.4 14.24 123.5 85.4
A soma do vetor eh '4584.06'

6) * (duasvezes)** Escreva uma função que retorne a quantidade de elementos que aparecem mais que 2 (duas) vezes em um vetor de tipo *float* e tamanho N (N definido via *define*).

```
#include <iostream>
using namespace std;

#define N 20



int duasvezes (float V[])
{
	int qtd[N],q=0;
	int i=0,j,k;
	int ja_verificado;
	while(i<N)
	{
		qtd[i]=1;
		j=0;
		ja_verificado=0;
		while(j<i)
		{
			if(V[i]==V[j])
				ja_verificado=1;
			j=j+1;
		}
		if(ja_verificado==0)
		{

			k=i+1;
			while(k<N)
			{
				if(V[i]==V[k])
					qtd[i]=qtd[i]+1;
				k=k+1;
			}
		}
	i=i+1;
	}
	i=0;

	//loop para contar a quantidade de posições que tem um numero >=2
	while(i<N)
	{
		if(qtd[i]>=2)
			q=q+1;
```

```
                i=i+1;
        }
        return q;
}

int main()
{
        float V[N],qtd;
        int i=0;
        cout<<"Entre com um vetor de 20 elementos:" <<endl;
        while(i < N)
        {
                cin>>V[i];
                i=i+1;
        }
        qtd=duasvezes(V);
        cout<<"A quantidade de elementos que aparecem mais de duas vezes
eh \'"<<qtd<<"\'" <<endl;
}
```

Exemplo de execução:

Entre com um vetor de 20 elementos:
41.8 54.4 133.5 123.45 41.54 85.3 123.45 85.4 85.6 32.5 85.4 985.7 52.4 52.4
123.45 3.3 214.4 14.24 123.5 85.4
A quantidade de elementos que aparecem mais de duas vezes eh ’3’

7) * (inverte) Dada uma sequência de N números, imprimí-la na ordem inversa à da leitura.

8) ** (verifpertence) Fazer um programa que leia e armazene em um vetor uma sequência de inteiros. Em seguida o programa deve ler uma sequência de inteiros informados pelo usuário e, para cada um deles, dizer se ele pertence ou não ao vetor armazenado previamente.

9) ** (histograma) Crie um programa que leia uma quantidade arbitrária N de números inteiros em um vetor e imprima, para cada número entre 0 e 100, a quantidade de vezes que este número ocorre no vetor lido inicialmente.

```
#include <iostream>
#define N 5
```

```
using namespace std;

void levetor (int V[])
{
	int i;
	i=0;
	while (i<N)
	{
		cin>>V[i];
		i=i+1;
	}
}

int ocorrencias (int V[], int x)
{
	int i, n;
	n=0;
	i=0;
	while (i<N)
	{
		if (V[i]==x)
			n=n+1;
		i=i+1;
	}
	return n;
}

int main()
{
	int V[N], x;
	cout<<"Entre com a sequencia de "<<N<<" inteiros:"<<endl;
	levetor(V);
	cout<<endl<<endl<<"HISTOGRAMA"<<endl;
	x=0;
	while (x<=100)
	{
		cout<<x<<": "<<ocorrencias(V,x)<<endl;
		x=x+1;
	}
}
```

Exemplo de execução:

```
./teste
Entre com a sequencia de 5 inteiros:
75 98 54 85 96
```

HISTOGRAMA
0: 0
1: 0
2: 0
3: 0
4: 0
5: 0
6: 0
7: 0
8: 0
9: 0
10: 0
11: 0
12: 0
13: 0
14: 0
15: 0
16: 0
17: 0
18: 0
19: 0
20: 0
21: 0
22: 0
23: 0
24: 0
25: 0
26: 0
27: 0
28: 0
29: 0
30: 0
31: 0
32: 0
33: 0
34: 0
35: 0
36: 0
37: 0
38: 0
39: 0
40: 0
41: 0
42: 0
43: 0
44: 0
45: 0
46: 0

47: 0
48: 0
49: 0
50: 0
51: 0
52: 0
53: 0
54: 1
55: 0
56: 0
57: 0
58: 0
59: 0
60: 0
61: 0
62: 0
63: 0
64: 0
65: 0
66: 0
67: 0
68: 0
69: 0
70: 0
71: 0
72: 0
73: 0
74: 0
75: 1
76: 0
77: 0
78: 0
79: 0
80: 0
81: 0
82: 0
83: 0
84: 0
85: 1
86: 0
87: 0
88: 0
89: 0
90: 0
91: 0
92: 0
93: 0
94: 0

```
95: 0
96: 1
97: 0
98: 1
99: 0
100: 0
```

10) ** (verifrepete) Escreva um programa que leia 2 sequências de 15 números cada e diga quantos números da segunda lista são iguais aos seus correspondentes da primeira lista.

```
#include <iostream>
#define N 5
using namespace std;

int repeticoes (int A[], int B[])
{
	int i, n;
	n=0;
	i=0;
	while (i<N)
	{
		if (A[i]==B[i])
			n=n+1;
		i=i+1;
	}
	return n;
}

void levetor (int V[])
{
	int i;
	i=0;
	while (i<N)
	{
		cin>>V[i];
		i=i+1;
	}
}

int main()
{
	int A[N], B[N];
	cout << "Entre com a sequencia 1 ("<< N << " elementos):" << endl;
	levetor(A);
	cout << "Entre com a sequencia 2("<< N << " elementos):" << endl;
```

```
	levetor(B);
	cout << "Repeticoes de numeros correspondentes = " << repeticoes(A,B) <<
endl;
}
```

Exemplo de execução:

```
Entre com a sequencia 1 (5 elementos):
1 3 5 7 9
Entre com a sequencia 2 (5 elementos):
0 3 0 7 9
Repeticoes de numeros correspondentes = 3
```

11) ** (verifigual) Fazer um programa que leia duas sequências de N números inteiros em dois vetores distintos, e verifique se os dois vetores são idênticos. A verificação da identidade entre os vetores deve ser feita por uma função denominada verifIdent() que recebe como parâmetros dois vetores de inteiros e retorna o valor 1 (um) se os vetores forem idênticos e 0 (zero) caso contrário.

```
#include <iostream>
#define N 5
using namespace std;

void levetor (int V[])
{
	int i;
	i=0;
	while (i<N)
	{
		cin>>V[i];
		i=i+1;
	}
}

int verifIdent (int A[], int B[])
{
	int i, n;
	n=0;
	i=0;
	while (i<N)
	{
		if (A[i]==B[i])
			n=n+1;
		i=i+1;
	}
```

```
	if (n==N)
		return 1;
	return 0; //Essa linha de comando so sera lida se a condicao if(n==N) não for
satisfeita
}


int main()
{
	int A[N], B[N], verifica;
	cout<<"Sequencia de inteiros 1:"<<endl;
	levetor(A);
	cout<<endl<<"Sequencia de inteiros 2:"<<endl;
	levetor(B);
	verifica=verifIdent(A, B);
	if (verifica==1)
		cout<<"As sequencias sao identicas";
	else //se verifica==0
		cout<<"As sequencias nao sao identicas";
	cout<<endl;
}
```

Exemplos de execução:

```
Sequencia de inteiros 1:
74 85 65 44 1

Sequencia de inteiros 2:
41 58 8 9 62
As sequencias nao sao identicas


Sequencia de inteiros 1:
4 5 8 9 52

Sequencia de inteiros 2:
4 5 8 9 52
As sequencias sao identicas
```

12) * (pertence)** Fazer um programa que leia duas sequências de inteiros, não necessariamente contendo a mesma quantidade de números (ou seja, a leitura do vetor encerra em zero, que não faz parte da sequência, ou no N-*ésimo* número da sequência), e diga se a segunda sequência está contida na primeira. Esta

verificação deve ser feita por uma função. O tamanho máximo das sequências deve estar definida no programa, via #define.

13) * (concatena)** Fazer um programa que leia duas sequências de inteiros, não necessariamente contendo a mesma quantidade de números (ou seja, a leitura do vetor encerra em zero, que não faz parte da sequência, ou no N-*ésimo* número da sequência), e construa um terceiro vetor, sem destruir os originais, que é a concatenação do primeiro com o segundo. Ou seja, se o primeiro vetor contiver os valores [7 3] e o segundo, [9 2], a concatenação dos dois conterá [7 3 9 2]. A concatenação deve ser feita por uma função. O tamanho máximo N das sequências deve estar definida no programa, via #define.

14) ** (intercala) Fazer um programa que leia dois conjuntos de inteiros contendo cada um a mesma quantidade de valores, e construa um terceiro vetor, sem destruir os originais, que é a intercalação do primeiro com o segundo. Ou seja, se o primeiro vetor contiver os valores [7 3] e o segundo, [9 2], a intercalação dos dois conterá [7 9 3 2]. A intercalação deve ser feita por uma função. O tamanho dos vetores deve estar definido no programa, via #define.

15) ** (polinomio) Dados n+1 valores reais, a_0, a_1, ..., a_n representando os coeficientes de um polinômio de grau n, calcular o valor do polinômio nos pontos X indicados pelo usuário, até o usuário entrar com o valor 999.99. O Valor do polinômio deve ser calculado pela função calc_pol(), que recebe 3 (três) parâmetros: um vetor de valores reais representando os coeficientes de um polinômio, um valor real representando o valor da variável X do polinômio, e uma referência para uma variável real que receberá o valor do polinômio no ponto X.

```
#include <iostream>
#include <cmath>
#define n 3
using namespace std;
```

```
void levetorC (float V[])
{
	int i;
	i=0;
	while (i<n+1)
	{
		cin>>V[i];
		i=i+1;
	}
}

void imprime_pol (float C[]) //Opcional
{
	int i;
	cout<<endl<<"p(x)= ";
	i=0;
	while (i<n+1)
	{
		cout<<C[i]<<"*x^"<<i<<" +  ";
		i=i+1;
	}
	cout<<endl<<endl;
}

void calc_pol (float C[], float x, float &p)
{
	int i;
	p=0;
	i=0;
	while (i<n+1)
	{
		p=p+C[i]*pow(x,i);
		i=i+1;
	}
}

int main()
{
	float C[n+1], x, p;
	cout<<"p(x) = a_0 + a_1x + a_2x2 + ..."<<endl<<endl<<"Entre com os
coeficientes, comecando pelo a_0"<<endl;
	levetorC(C);
	imprime_pol (C); //Opcional
	cout<<endl<<"Entre com x: ";
	cin>>x;
	while (x!=999.99)
	{
```

```
                calc_pol (C, x, p);
                cout<<”p(“<<x<<”) = “<<p<<endl<<endl<<”Entre com x: “;
                cin>>x;
        }
        cout<<endl;
}

#include <iostream>
#include <cmath>
#define n 3
using namespace std;

void levetorC (float V[])
{
        int i;
        i=0;
        while (i<n+1)
        {
                cin>>V[i];
                i=i+1;
        }
}

void imprime_pol (float C[]) //Opcional
{
        int i;
        cout<<endl<<"p(x)= ";
        i=0;
        while (i<n+1)
        {
                cout<<C[i]<<"*x^"<<i<<" +  ";
                i=i+1;
        }
        cout<<endl<<endl;
}

void calc_pol (float C[], float x, float &p)
{
        int i;
        p=0;
        i=0;
        while (i<n+1)
        {
                p=p+C[i]*pow(x,i);
                i=i+1;
        }
}
```

```
int main()
{
	float C[n+1], x, p;
	cout<<"p(x) = a_0 + a_1x + a_2x2 + ..."<<endl<<endl<<"Entre com os
coeficientes, comecando pelo a_0"<<endl;
	levetorC(C);
	imprime_pol (C); //Opcional
	cout<<endl<<"Entre com x: ";
	cin>>x;
	while (x!=999.99)
	{
		calc_pol (C, x, p);
		cout<<"p("<<x<<") = "<<p<<endl<<endl<<"Entre com x: ";
		cin>>x;
	}
	cout<<endl;
}
```

Exemplos de execução:

```
p(x) = a_0 + a_1x + a_2x2 + ...

Entre com os coeficientes, comecando pelo a_0
5.8 41.2 32.8 126

p(x)= 5.8*x^0 +  41.2*x^1 +  32.8*x^2 +  126*x^3 +


Entre com x: 2
p(2) = 1227.4

Entre com x: 6
p(6) = 28649.8

Entre com x: 2.4
p(2.4) = 2035.43
```

16) * (ordenaint)** Fazer um programa que leia duas sequências de inteiros, não necessariamente contendo a mesma quantidade de números, ordene-as e, em seguida, imprima todos os números ordenados em ordem crescente. Sugestão: Faça a primeira sequência com tamanho 4 e a segunda sequência com tamanho 5. Exemplo:

```
V1: 7 3 2 6
```

V2: 5 1 8 4 9
Saída: 1 2 3 4 5 6 7 8 9

17) * (escalar) Dados dois vetores *X* e *Y* de 20 posições cada, determinar o produto escalar entre os dois vetores. O produto escalar entre dois vetores é dado por:

$$X_0Y_0 + X_1Y_1 + X_2Y_2 + \ldots + X_{19}Y_{19}$$

18) * (qtdistintos)** Dada uma sequência de N números, determinar quantos números distintos compõem a sequência e o número de vezes que cada um deles ocorre na mesma. Exemplo:

N=5
1 2 3 2 3
a sequência tem três números distintos, 1, 2 e 3. Ocorrências:
1 1 vez
2 2 vezes
3 2 vezes

19) * (localizaX) Escreva uma função que recebe um vetor, seu tamanho e um número *X*. Mostre a posição de cada elemento igual a *X* deste vetor.

20) * (substnulos) Escreva uma função que recebe um vetor e seu tamanho. Substitua todos os valores nulos (0) do vetor por 1.

21) * (trocaelems)** Escreva uma função que recebe um vetor e seu tamanho. A função deve inverter a ordem dos elementos do vetor (trocar o 1º elemento com o último, o 2º elemento com o penúltimo, etc.).

22) * Crie um programa que leia dez caracteres, separados um do outro por um espaço em branco, correspondentes a uma palavra, e imprima os caracteres na ordem inversa da ordem lida.

23) ** Crie um programa que leia vinte números reais e imprima a sua média e desvio-padrão.

24) * Crie um programa que leia uma palavra com até 20 caracteres, com um espaço em branco entre cada par de caracteres. Em seguida, para cada letra da palavra, altere a letra para a letra seguinte no alfabeto. Após alterar a palavra, imprima a palavra encontrada.

25) *** Crie um programa que leia duas palavras com até 20 caracteres cada. Em seguida, informe qual das duas palavras seria impressa antes no dicionário. Exemplos: ABACO é impresso antes de BALEIA. CACHORRO é impresso antes de CURITIBA. JOAO é impresso antes de JOSE. JAPAO é impresso antes de JAPONES. E assim por diante.

26) *** Crie um programa que leia uma palavra com até 100 caracteres, sempre com um espaço em branco entre dois caracteres quaisquer. Em seguida, leia uma segunda palavra com até 10 caracteres, também com os devidos espaços em branco. O programa deve imprimir a posição em que a segunda palavra foi encontrada dentro da primeira. Exemplo: se a primeira palavra for OTORRINOLARINGOLOGISTA, e a segunda palavra for GOL, então deve imprimir 13 (GOL é encontrado na posição 13 na palavra OTORRINOLARINGOLOGISTA). Informar ao usuário o caso em que a segunda palavra não está contida na primeira (ou porque a segunda palavra é maior ou porque não tem letras coincidentes).

EXERCÍCIOS: 7 – MATRIZES

1) *(preencheprod) Faça um programa que preenche uma matriz 15x9 com o produto dos índices de cada posição.

```
#include <iostream>
#define N 9
#define M 15
using namespace std;

int imprime_matriz(int matriz[N][M])
{
	for (int i = 0; i < N ; i++)
	{
		for (int j = 0; j < N; j++)
			cout << matriz[i][j] << " ";
		cout <<endl;
	}
}
int main()
{
	int matriz[N][M];
	int i, j;
	i = 0;
	while (i < N)
	{
		j = 0;
		while (j < N)
		{
			matriz[i][j]=i*j;
			j++;
		}
		i++;
	}
	imprime_matriz(matriz);
	cout << endl;
	return 0;
}
```

Exemplo de execução:

```
0 0 0 0 0 0 0 0 0
0 1 2 3 4 5 6 7 8
0 2 4 6 8 10 12 14 16
0 3 6 9 12 15 18 21 24
```

0 4 8 12 16 20 24 28 32
0 5 10 15 20 25 30 35 40
0 6 12 18 24 30 36 42 48
0 7 14 21 28 35 42 49 56
0 8 16 24 32 40 48 56 64

2) * (quadradomagico)** Dizemos que uma matriz quadrada inteira é um quadrado mágico se a soma dos elementos de cada linha, a soma dos elementos de cada coluna e a soma dos elementos das diagonais principal e secundária são todos iguais. Exemplo:

8	0	7
4	5	6
3	10	2

é um quadrado mágico pois

8+0+7 = 4+5+6 = 3+10+2 = 8+4+3 = 0+5+10 = 7+6+2 = 8+5+2 = 3+5+7

Dada uma matriz quadrada A (n x m), verificar se A é um quadrado mágico.

```
#include <iostream>
#define N 3
#define M 3
using namespace std;
bool quadradomagico(int matriz[N][M])
{
	int i, j;
	i = 0;
	int soma1 = 0;
	int soma2 = 0;
	// soma primeira linha

	for (j = 0; j< M; j++)
	{
		soma1 += matriz[i][j];
	}
	// confere com soma das outras linhas
	for (i = 1; i < N; i++)
	{
		soma2 = 0;
		for (j = 0; j < M; j++)
		{
```

```
                soma2 += matriz[i][j];
        }

        if (soma1 != soma2)
                return false;
}

// confere soma das colunas

for (i=0; i < N; i++)
{
        soma2 = 0;
        for (j = 0; j< M; j++)
        {
                soma2+= matriz[j][i];
        }

        if (soma1 != soma2)
                return false;
}

// confere diagonal principal
soma2 = 0;
for (i=0; i < N; i++)
{
        soma2+= matriz[i][i];
}
if (soma1 != soma2)
        return false;

// confere diagonal secundaria
soma2 = 0;
for (i=0; i < N; i++)
{
        soma2+= matriz[i][N - i - 1];
}
if (soma1 != soma2)
{
        cout << soma2 << endl;
        return false;
}

// se todas as somas forem iguais, chegou aqui
// logo, eh uma quadrado magico
return true;
```

```
}

int main()
{
	int matriz[N][M]= {8, 0, 7, 4, 5, 6, 3, 10, 2};
	int matriz2[N][M]={9, 0, 7, 3, 5, 6, 3, 10, 2};
	if(quadradomagico(matriz))
		cout << "Matriz 1 eh um quadrado magico" << endl;
	else
		cout << "Matriz 1 nao eh um quadrado magico" << endl;
	if(quadradomagico(matriz2))
		cout << "Matriz 2 eh um quadrado magico" << endl;
	else
		cout << "Matriz 2 nao eh um quadrado magico" << endl;
	return 0;
}
```

Exemplo de execução:

```
Matriz 1 eh um quadrado magico
Matriz 2 nao eh um quadrado magico
```

3) *(somamat) Dadas duas matrizes reais A (m x n) e B (m x n), fazer um programa que calcula a soma de A com B. A soma deve ser efetuada por uma função que recebe 3 (três) matrizes: a primeira recebe o resultado da soma das outras duas.

```
#include <iostream>

#define N 3
#define M 3

using namespace std;

int imprime_matriz(int matriz[][M])
{
	for (int i = 0; i < N ; i++)
	{
		for (int j = 0; j < M; j++)
		{
			cout << matriz[i][j] << "\t";
		}
		cout <<endl;
	}
```

```
}

int soma_matrizes(int A[][M], int B[][M], int C[][M])
{
	for (int i = 0; i < N ; i++)
	{
		for (int j = 0; j < M; j++)
		{
			C[i][j] = A[i][j] + B[i][j];
		}
		cout <<endl;
	}
		return 0;
}

int main()
{
	int A[N][M] = {0, 1, 2, 3, 4, 5, 6, 7, 8};
	int B[N][M] = {3, 5, 7, 9, 11, 13, 15, 17, 19};
	int C[N][M];
	cout << "Matriz A:" << endl;;
	imprime_matriz(A);
	cout << "Matriz B:" << endl;
	imprime_matriz(B);
	soma_matrizes(A, B, C);
	cout << "A + B = C:" <<endl;
	imprime_matriz(C);
	return 0;
}
```

Exemplo de execução:

```
Matriz A:
0	1	2
3	4	5
6	7	8
Matriz B:
3	5	7
9	11	13
15	17	19



A + B = C:
3	6	9
12	15	18
21	24	27
```

4) **(triangpascal) Um coeficiente binomial, $C(n,k)$, representa o número de possíveis combinações de N elementos tomados K a K. Um Triângulo de Pascal, uma homenagem ao grande matemático Blaise Pascal, é uma tabela de valores de coeficientes combinatoriais para pequenos valores de N e K. Os números que não são mostrados na tabela têm valor zero.

Este triângulo pode ser construído automaticamente usando-se uma propriedade conhecida dos coeficientes binomiais, denominada fórmula da adição: $C(r,k) = C(r-1,k) + C(r-1,k-1)$. Ou seja, cada elemento do triângulo é a soma de dois elementos da linha anterior, um da mesma coluna e um da coluna anterior. Veja um exemplo de um triângulo de Pascal com 7 linhas, com uma indicação de como obter os elementos:

```
1
1       1
1       2       1
1       3       3       1
1       4       6       4       1
1       5       10      10      5       1
1       6       15      20      15      6       1
```

Faça um programa para gerar um triângulo de PASCAL de N linhas em uma matriz de tamanho NxN.

```
#include <iostream>
#define N 7
using namespace std;

int imprime_pasc(int matriz[N][N])
{
        for (int i = 0; i < N ; i++)
        {
                for (int j = 0; j <= i; j++)
                {
                        cout << matriz[i][j] << "\t";
                }
                cout <<endl;
```

```
        }
}

int gerapascal(int matriz[N][N])
{
        // inicializa primeiros elementos da matriz
        matriz[0][0] = 1;
        matriz[1][0] = 1;
        matriz[1][1] = 1;
        int i = 2;
        while (i < N)
        {
                int j = 0;
                while (j <= i)
                {
                        if (j == 0)
                                matriz[i][j]=matriz[i-1][j];
                        else if (j == i)
                                matriz[i][j]=matriz[i-1][j-1];
                        else
                                matriz[i][j]=matriz[i-1][j-1] + matriz[i-1][j];
                        j++;
                }
                i++;
        }
}

int main()
{
        int matriz[N][N];
        gerapascal(matriz);
        imprime_pasc(matriz);
        return 0;
}
```

Exemplo de execução:

```
1
1       1
1       2       1
1       3       3       1
1       4       6       4       1
1       5       10      10      5       1
1       6       15      20      15      6       1
```

5) *(maiorelem) Fazer um programa que calcula o maior elemento de cada linha de uma matriz e coloca estes valores em um vetor, onde cada n-ésimo elemento deste corresponde ao maior elemento da n-ésima linha da matriz.

```
#include <iostream>
#define N 5
using namespace std;

int imprime_matriz(int matriz[N][N])
{
	for (int i = 0; i < N ; i++)
	{
		for (int j = 0; j < N; j++)
		{
			cout << matriz[i][j] << "\t";
		}
		cout <<endl;
	}
}

int trocaelems(int matriz[N][N])
{
	int i, j, maior, aux;
	i = 0;
	while (i < N)
	{
		j = 0;
		maior = 0;
		while (j < N)
		{
			if (matriz[i][j] > matriz[i][maior])
			maior = j;
			j++;
		}
	aux = matriz[i][0];
	matriz[i][0]=matriz[i][maior];
	matriz[i][maior]=aux;
	i++;
	}
}

int main()
{
	int matriz[N][N];
	cout << "Entre com uma matriz 5x5 (25 numeros)" << endl;
	int i = 0;
```

```
        int j;
        while (i < N)
        {
                j = 0;
                while (j < N)
                {
                        cin >> matriz[i][j];
                        j++;
                }
                i++;
        }
        cout << endl;
        imprime_matriz(matriz);
        cout << endl;
        trocaelems(matriz);
        imprime_matriz(matriz);
        return 0;
}
```

Exemplo de execução:

```
Entre com uma matriz 5x5 (25 numeros)
47 58 462 214 36
478 5 62 34 1
41 85 96 74 3
41 25 47 74 5
1 4 78 99 65 99

47      58      462     214     36
478     5       62      34      1
41      85      96      74      3
41      25      47      74      5
1       4       78      99      65

462     58      47      214     36
478     5       62      34      1
96      85      41      74      3
74      25      47      41      5
99      4       78      1       65
```

6) * Crie um programa que leia uma matriz de 3 linhas e 4 colunas e imprima a mesma matriz lida.

7) * Crie um programa que leia uma matriz de 3 linhas e 4 colunas imprima a matriz transposta da matriz lida.

8) * Crie um programa que leia uma matriz de 4 linhas e 4 colunas e imprima os elementos da diagonal principal da matriz lida.

9) ** Crie um programa que leia uma matriz de 4 linhas e 4 colunas e imprima os elementos da diagonal secundária da matriz lida.

10) ** (prodmatvet) Dada uma matriz de valores to tipo real A com m linhas e n colunas e um vetor real V com n elementos, determinar o produto de A por V.

11) * (prodmat)** Dadas duas matrizes reais A (m x n) e B (n x p) calcular o produto de A por B.

12) * (repetemat)** Dada uma matriz real A (m x n), verificar se existem elementos repetidos em A.

13) ** (prndiagonais) Crie um programa que leia uma matriz de 10 linhas e 10 colunas e imprima em linhas distintas os elementos da diagonal principal e da diagonal secundária da matriz lida.

14) * (somamat) Dadas duas matrizes reais A (m x n) e B (m x n), fazer um programa que calcula a soma de A com B. A soma deve ser efetuada por uma função que recebe 3 (três) matrizes: a primeira recebe o resultado da soma das outras duas.

15) ** (matpermuta) Dizemos que uma matriz inteira A (n x n) é uma matriz de permutação se em cada linha e em cada coluna houver n-1 elementos nulos e um único elemento igual a 1. Dada uma matriz inteira A (n x n) verificar se A é de permutação. Exemplos:

0 1 0 0
0 0 1 0

1 0 0 0
0 0 0 1

é de permutação, enquanto que:

0 1 0 0
0 0 1 0
1 0 0 0
0 0 0 2

não é.

16) ** (lincolnulas) Dada uma matriz A (n x m) imprimir o número de linhas e o número de colunas nulas da matriz. Exemplo:

0 0 0 0
1 0 2 2
4 0 5 6
0 0 0 0

tem duas linhas nulas e uma coluna nulas

17) * (custotransp)** Os elementos M[i,j] de uma matriz M (n x n) representam os custos de transporte da cidade i para a cidade j. Dados n itinerários, cada um com k cidades, calcular o custo total para cada itinerário. Exemplo:

4 1 2 3
5 2 1 40 0
2 1 3 8
7 1 2 5

O custo do itinerário 1 4 2 4 4 3 2 1 é:
M[1,4]+*M*[4,2]+*M*[2,4]+*M*[4,4]+*M*[4,3]+*M*[3,2]+*M*[2,1] = 3+1+400+5+2+1+5 = 417.

EXERCÍCIOS: 8 – QUESTÕES DE PROVAS

1) Escreva um programa em C++ para ler 3 números inteiros de no máximo 2 dígitos, e verificar se a soma deles é maior que 100. Caso o valor da soma seja maior que 100, o programa deve imprimir a mensagem "limite indisponível". Caso contrário, a mensagem "limite permitido" deve ser exibida. O programa não deve aceitar valores com mais que 2 dígitos.

Exemplo de execução:
Indique 3 números inteiros: 2 20 12
limite permitido

Outro exemplo de execução do programa:
Indique 3 números inteiros: 2 90 12
limite indisponível

Outro exemplo de execução do programa:
Indique 3 números inteiros: 2 190 12
Valores inválidos

2) Escreva um programa em linguagem C++ que leia, a partir do teclado, a data de nascimento (dia, mês e ano) de uma pessoa e determine quantos anos completos ela terá no dia 25/03/2014.

Exemplo de execução:
Data nascimento: 17 05 1988
Idade: 25 anos

3) Uma empresa decidiu dar uma gratificação de Natal a seus funcionários. Esta gratificação é calculada com base no número de horas extras trabalhadas e o número de horas de faltas ao trabalho. A fórmula para se calcular as horas premiadas é HorasPremiadas = HorasExtras − 2 3 × HorasFaltas. O prêmio é distribuído segundo a tabela a seguir:

Horas Premiadas	Prêmio (R$)
até 10 horas (inclusive)	10,00
de 11 a 20 horas (inclusive)	20,00

de 21 a 30 horas (inclusive)	30,00
de 31 a 40 horas (inclusive)	40,00
acima de 40 horas	50,00

Faça um programa em C++ que pergunte ao usuário o número de horas extras trabalhadas e o número de horas em que faltou ao trabalho, e então mostre na tela o prêmio em reais a ser recebido.

Exemplo de execução:
Horas extras: 30
Horas de faltas: 3
Premio: R$ 30.00

4) Um programa para gerenciar as contas de um banco deve possuir algum mecanismo para calcular o saldo dos clientes após as movimentações diárias. Escreva um programa em C++ que receba o valor da quantia em conta e após, o código da operação realizada e o valor associado. O programa deve retornar o saldo em conta. Se o saldo for negativo, ele deve imprimir mensagem informando que o cliente entrou no cheque especial. As operações permitidas são:

100 depósito em dinheiro
101 depósito em cheque
102 depósito de outra conta
200 saque em dinheiro
201 saque com cheque
202 saque para outra agência

Exemplo de execução:
Saldo: 2000 202 1000
Saldo em conta: 1000

Outro exemplo de execução:
Saldo: 1000 200 1500
Você entrou no Cheque Especial !!

5) Um atleta treina em uma pista de corrida circular de 400 metros. Ao final do treino, com base na quantidade de voltas dadas e no tempo total do treino (em horas, minutos e segundos), ele quer saber:

• O percurso total do treino (em quilômetros);
• Otempomédioporquilômetro(emhoras,minutos e segundos);

• O tipo de atleta, de acordo com o tempo médio por quilômetro:
– Atleta de competição, se o tempo médio está no intervalo entre 0 e 4 minutos (inclusive);
– Atleta participante, se o tempo médio está no intervalo acima de 4 e abaixo de 7 minutos (7 inclusive);
– Participante de caminhada, se o tempo médio é acima de 7 minutos.

Faça um programa em C++ que auxilie o atleta a ter essas respostas. Para o tempo, se a sua solução tiver valores abaixo de segundos, ignore a parte decimal / fracionária após os segundos.

Exemplo de execução:
No. voltas: 13
Tempo treino(hh mm ss): 0 55 23
Percurso total (km): 5.2
Tempo médio/km (hh mm ss): 0 10 39
Tipo: Participante de caminhada

Outro exemplo de execução:
No. voltas: 13
Tempo treino(hh mm ss): 0 20 30
Percurso total (km): 5.2
Tempo médio/km (hh mm ss): 0 3 57
Tipo: Atleta de competição

6) Faça um programa em C++ que leia 4 números inteiros Ha, Ma, Hd e Md, onde Ha : Ma representa o horário atual (Ha horas e Ma minutos) e Hd : Md representa o horário programado em um despertador (Hd horas e Md minutos). Seu programa deve calcular e mostrar quanto tempo resta até que o alarme desperte (quantas horas e quantos minutos). Se os horários forem idênticos, seu programa deve indicar que o alarme está despertando e, se o horário programado for anterior ao horário atual, seu programa deve indicar que o alarme já despertou. Considere que os valores Ha e Hd estão sempre entre 0 e 23 e que os valores Ma e Md estão sempre entre 0 e 59. Além disso, considere que o horário programado no despertador é referente ao dia atual e nunca ao dia seguinte.

Exemplo de execução:
10 12 14 15
4 horas e 3 minutos para despertar

Outro exemplo de execução:

16 34 22 06
5 horas e 32 minutos para despertar

Outro exemplo de execução:
23 42 03 06
Alarme já despertou!

Outro exemplo de execução:
09 42 09 42
Alarme está despertando!

Outro exemplo de execução:
13 45 13 40
Alarme já despertou!

7) Escreva um programa em C++ que lê dois números inteiros A e B do teclado e imprime as mensagens Nulo, Sinal negativo ou Sinal positivo, conforme o resultado da multiplicação de A por B, SEM REALIZAR A OPERAÇÃO.

Exemplo de execução:
10 12
Sinal positivo

Outro exemplo de execução:
16 -34
Sinal negativo

Outro exemplo de execução:
0 -34
Nulo

8) Você foi contratado por uma nova república que necessita fornecer um programa escrito em C++ para realizar o ajuste annual de rendimentos pessoa física. O programa deve ler:

• RBA: Renda bruta anual;
• NDP: Número de dependentes;
• IPG: Imposto pago no ano.

A base de cálculo (BC) é dada por RBA − NDP × 1500,00. O número máximo de dependentes dedutíveis é 4. Para o cálculo do imposto devido você deve utilizar a

tabela progressiva abaixo para definir o imposto devido (IBC), que é calculado aplicando a porcentagem indicada (alíquota) sobre a Base de Cálculo:

BC	**Alíquota**
até 15.000,00	0% (isento)
de 15.000,01 até 35.000,00	7,5%
de 35.000,01 até 280.000,00	15,0%
acima de 280.000,00	60%

Com estes dados você deve calcular o imposto devido e, descontando deste o imposto que já foi pago no ano (IPG), informar ao usuário:

• Valor final do imposto a pagar
• Valor final da restituição a receber
• Mensagem indicando que usuário tem imposto quitado.

Exemplo de execução:
RBA: 55000.00
NDP: 3
IPG: 7230.00
Imposto a pagar: R$ 345.00

Outro exemplo de execução:
RBA: 35000.00
NDP: 2
IPG: 10230.00
Restituição a receber: R$ 7830.00

9) Dada a atual crise hídrica do país, as pessoas começaram a construir reservatórios para armazenar água em suas propriedades. Faça um programa em C++ que auxilie os utilizadores do reservatório a controlarem seu consumo. Obtenha do teclado as dimensões de um reservatório (altura, largura e comprimento, em centímetros) e o consumo médio diário dos utilizadores do reservatório (em litros / dia). Assuma que o reservatório esteja cheio, tenha formato cúbico e informe:

(a) A capacidade total do reservatório, em litros;
(b) A autonomia do reservatório, em dias;
(c) A classificação do consumo, de acordo com a quantidade de dias de autonomia:
Consumo elevado, se a autonomia for menor que 2 dias;
Consumo moderado, se a autonomia estiver entre 2 e 7 dias;

Consumo reduzido, se a autonomia maior que 7 dias.

Obs.: Considere que cada litro equivale a 1000 cm3 ou 1 dm3.

Exemplo de execução:
Altura do reservatorio (cm): 100
Largura do reservatorio (cm): 100
Comprimento do reservatorio (cm): 100
Consumo medio diario (litros/dia): 400
Capacidade reservatorio: 1000 litros
Autonomia reservatorio: 2.5 dias
Consumo moderado

Outro exemplo de execução:
Altura do reservatorio (cm): 50
Largura do reservatorio (cm): 100
Comprimento do reservatorio (cm): 150
Consumo medio diario (litros/dia): 400
Capacidade reservatorio: 750 litros
Autonomia reservatorio: 1.875 dias
Consumo elevado

10) Vamos chamar de crescente um número natural n = d1d2 ...dk, cujos dígitos di estão em ordem crescente, isto é, tal que d1 < d2 < ⋯ < dk. Faça um programa em C++ que leia um número inteiro e positivo n, de 3 dígitos, e verifique se n é crescente. Seu programa deve também verificar se n possui exatamente 3 dígitos e imprimir mensagens adequadas em cada caso.

Exemplo de execução:
Entre com um inteiro positivo: 3416
valor invalido.

Outro exemplo de execução:
Entre com um inteiro positivo: 152
152 nao eh crescente.

Outro exemplo de execução:
Entre com um inteiro positivo: 378
378 eh crescente.

11) Escreva um programa em C++ que receba dois valores representando a concentração inicial de duas bactérias e outros dois valores representando as duas

respectivas taxas diárias de crescimento das culturas de bactérias. O programa deve calcular e mostrar o número de dias necessários para que a cultura da primeira bactéria ultrapasse a da segunda. Assumir que o usuário SEMPRE informa a taxa de crescimento da primeira bactéria maior que a da segunda e que ele SEMPRE informa valores de taxas entre 0 (zero) e 1 (um).

Exemplo de execução:
Concentracao inicial da bacteria 1: 40
Taxa de crescimento da bacteria 1 (entre 0 e 1): 0.5
Concentracao inicial da bacteria 2: 100
Taxa de crescimento da bacteria 2 (entre 0 e 1): 0.1
A 1a bacteria ultrapassa a 2a em 3 dia(s)

12) Escreva um programa em linguagem C++ que leia, a partir do teclado, uma quantidade arbitrária de pares de números reais, sendo um par de Zeros a identificação do final dos pares, e determine a média dos produtos positivos e dos produtos negativos destes pares.

Exemplo de execução:
73.5 9.8
17.3 -25.3
-8.7 19.5
-22.9 -32.7
91.2 19.2
0 0

Media dos produtos positivos: 1073.39
Media dos produtos negativos: -303.67

13) Escrever um programa em C++ que leia do teclado dois números inteiros positivos com 3 algarismos, respectivamente os limites inferior e superior de um intervalo de números inteiros. Validado o intervalo (vide exemplo de execução), pede-se que o programa liste e conte os números inteiros entre os dois limites (incluindo estes) em que o dígito da dezena seja 3, 5 ou 7.

Exemplo de execução:
Informe limites (inferior e superior): 128 115
Limites invalidos

Outro exemplo de execução:
Informe limites (inferior e superior): 23 12000
Limites invalidos.

Outro exemplo de execução:
Informe limites (inferior e superior): 110 150
130 131 132 133 134 135 136 137 138 139 150
Nos. Listados: 11

14) Escreva um programa em C++ que leia dois números inteiros, x, e n, sendo n não-negativo, e que calcule o valor de xn. Não é permitido o uso da função float pow(float x, float y) ou de qualquer outra função pré-existente.

Exemplo de execução:
Digite valores x e n: 3 4
O valor de 3^4 é: 81

Outro exemplo de execução:
Digite valores x e n: -3 3
O valor de -3^3 é: -27

Outro exemplo de execução:
Digite valores x e n: 3 -3
O expoente deve ser positivo.

Outro exemplo de execução:
Digite valores x e n: 3 0
O valor de 3^0 é: 1

15) Escreva um programa em C++ que leia do teclado conjuntos de três medidas a, b, e c (reais e positivas) correspondentes aos lados de triângulos. Para cada tripla lida, o programa deverá verificar e informar se é ou não realmente possível a montagem de um triângulo. Sendo possível, o perímetro do triângulo deverá ser calculado e mostrado no monitor de vídeo. Não sendo possível, calcular e mostrar a dimensão mínima ACIMA DA QUAL a adição desta a algum dos lados menores informados possibilitará a formação de um triângulo. O conjunto de triplas se encera com a informação da primeira medida nula ou negativa; esta tripla não faz parte do conjunto de dados a processar. Ao final do programa deverão ser mostrados o total de triângulos possíveis de montagem, seu perímetro médio, o total de triângulos não possíveis de montagem, a média dos incrementos faltantes e o total de triplas lidas.

Observação: seu programa não precisa validar se as medidas informadas são reais e positivas. Lembre-se que para formar um triângulo, a soma de dois lados quaisquer é sempre maior que a medida do terceiro lado, isto é, dados a, b, e c, eles formam um triângulo se e somente se a + b > c,b + c > a,a + c > b.

Exemplo de execução:
Medidas: 3 4 5
Formam um triângulo.
Perímetro: 12
Medidas: 10 2 4
Não formam um triângulo.
Incremento mínimo > 4
Medidas: 2.5 3 4.5
Formam um triângulo.
Perímetro: 10
Medidas: 6 6 5
Formam um triângulo.
Perímetro: 17
Medidas: 1 1 7
Não formam um triângulo.
Incremento mínimo > 5
Medidas: -3 1 2
Triângulos montados: 3
Perímetro médio: 13
Triângulos não montados: 2
Média dos incrementos faltantes: 4.5
Total de triplas: 5

16) Faça um programa em C++ que leia um conjunto de valores inteiros n1,n2,...,nd representando as faces obtidas em d lançamentos de um dado de seis faces e imprime uma mensagem indicando se o dado é regular ou se está viciado. Considere que todo valor lido ni está entre 1 e 6, isto é, $1 \leq ni \leq 6$, e que o final do conjunto de valores é indicado quando o usuário digitar o valor 0 (zero), que não deve ser contado como um lançamento do dado. Além disso, considere que o dado está viciado se o número de ocorrências de alguma de suas faces for maior que o dobro do valor esperado, definido pelo número total de lançamentos multiplicado pela probabilidade de ocorrência da respectiva face. Isto é, Dado viciado $\Longleftrightarrow$ Ocorrências de alguma das faces > 2×Número total de lançamentos× 1/6

Exemplo de execução: 1 1 2 5 4 6 0
Dado regular

Outro exemplo de execução: 6 4 3 4 1 4 4 2 0
Dado viciado

17) Faça um programa em C++ que leia um conjunto de pares de valores inteiros positivos (p1,n1),(p2,n2),⋯ ,(pk,nk) onde pi representa uma praça de pedágio saindo de Curitiba e ni representa a quantidade de carros que passaram nas últimas 24 horas por essa estação de pedágio pi. Foram avaliadas 5 praças de pedágios saindo de Curitiba. Considere que todos os valores lidos são inteiros e positivos, e que o valor lido pi está entre 1 e 5, isto é, 1 6 pi 6 5, e que o final do conjunto de valores é indicado quando o usuário digitar os valores 0 0 (zero zero). No final o programa deverá informar (a) A quantidade total de veículos que passou em cada Praça de Pedágio, (b) A quantidade total de carros que passaram pelas praças de pedágio, e (c) A quantidade média de veículos por praça de pedágio. Cada praça de pedágio pi pode receber vários lançamentos de ni que deverão ser somados.

Exemplo de execução:
Informe as praças e quantidades:
1 1250
2 3020
4 3018
5 4109
3 3298
2 765
1 435
2 3278
0 0
Praça de Pedágio 1: 1685 veículos
Praça de Pedágio 2: 7063 veículos
Praça de Pedágio 3: 3298 veículos
Praça de Pedágio 4: 3018 veículos
Praça de Pedágio 5: 4109 veículos
Quantidade total: 19173 veículos
Média por Praça de Pedágio: 3834.60 veículos

18) Escreva um programa em C++ para calcular e exibir os tempos gastos e alturas percorridas por uma lesma em suas escaladas em arranha-céus. Para isso, o

usuário deverá informar a altura total a ser escalada (Alt), a quantidade de metros que a lesma consegue escalar por turno de 12 horas de subida (Vsubida) e a quantidade de metros que ela escorrega por turno de 12 h de descanso (Vdescida). Os valores de entrada para Alt e Vsubida devem ser positivos e não-nulos, e o valor para Vdescida deve estar no intervalo [0,Vsubida). Caso estas restrições não sejam satisfeitas, o programa deve terminar sem efetuar qualquer cálculo, exibindo uma mensagem indicando o fato transgressor. A escalada deverá ser descrita como nos exemplos a seguir.

Exemplo de execução:
Altura total a escalar (m): 33
Escalada de subida (m / 12h): 10
Escorregamento de descanso (m / 12h): 2
Desenvolvimento da escalada:
0h / 0m
12h / 10m
24h / 8m
36h / 18m
48h / 16m
60h / 26m
72h / 24m
82,8h / 33m <Objetivo atingido>
Obs.: 82,8h: 33m - 24m = 9m / 10m = 0,9 x 12h = 10,8h + 72h = 82,8h

Outro exemplo de execução:
Altura total a escalar (m): 5
Escalada de subida (m / 12h): 10
Escorregamento de descanso (m / 12h): 2
Desenvolvimento da escalada: 0h / 0m 6h / 5m <Objetivo atingido>
Obs.: 6h: 5m - 0m = 5m / 10m = 0,5 x 12h = 6h + 0h = 6h

19) Escreva um programa em C++ que leia do teclado uma sequência de números inteiros até que seja lido um número que seja o dobro ou a metade do anterior. O programa deve imprimir como saída os seguintes valores:

(a) A quantidade de números lidos;
(b) A soma dos números lidos;
(c) Os dois valores que forçaram a parada do programa.

Exemplo de execução:
4 3 15 8 16
5 46 8 16

Outro exemplo de execução:
-424 646 438 892 964 384 192
7 3092 384 192

20) O licenciamento anual de um veículo deve ser pago de acordo com o último dígito da sua placa, nos meses previstos na tabela abaixo:

Final da placa	Mês de renovação
1	Abril
2	Maio
3	Junho
4	Julho
5 e 6	Agosto
7	Setembro
8	Outubro
9	Novembro
0	Dezembro

Uma locadora de veículos precisa saber quantos licenciamentos terá de pagar a cada mês, de acordo com as placas na sua frota. Escreva um programa em C++ que leia uma sequência de placas de automóveis (-1 indica o término da sequência), cada placa representada apenas pelos 4 dígitos, e que determine (a) a quantidade de licenciamentos que deverá ser paga a cada mês, (b) a quantidade de carros na frota da locadora e (c) a média mensal de licenciamentos que deverão ser pagos.

Exemplo de execução:
Informe as placas dos veículos:
4444 2231 4451 3422 1230 3211 2376 6789 9820 3425 -1
Total de Abril: 3
Total de Maio: 1
Total de Junho: 0
Total de Julho: 1
Total de Agosto: 2
Total de Setembro: 0
Total de Outubro: 0
Total de Novembro: 1
Total de Dezembro: 2

Tamanho da frota: 10 veículos
Média mensal de licenciamentos pagos: 1,1

21) Em época de eleição, sua tarefa é escrever um programa em C++ para simular uma urna eletrônica. Na eleição para presidente, existem 3 candidatos que utilizam respectivamente os códigos: 10, 11, 12. Adicionalmente é utilizado o código 1 para voto nulo e 2 para voto em branco. Seu programa deve receber votos até que o valor 0 seja informado. Ao final, seu programa deve mostrar na tela o total de votos de cada candidato, nulos e em branco e o total de votos na eleição.

Exemplo de execução:
Informe seu voto: 10
Informe seu voto: 11
Informe seu voto: 11
Informe seu voto: 12
Informe seu voto: 11
Informe seu voto: 10
Informe seu voto: 1
Informe seu voto: 2
Informe seu voto: 2
Informe seu voto: 0
Candidato 1: 2 votos
Candidato 2: 3 votos
Candidato 3: 1 voto
Votos Nulos: 1
Votos em branco: 2
Total de votos: 9

22) Escreva um programa em C++ que deverá ler do teclado um conjunto de valores reais positivos, até que seja digitado um valor negativo. Devem ser determinados e impressos na tela o valor médio destes valores e o quadrado da diferença entre a soma dos valores e o valor médio calculado. Nestes cálculos, o valor negativo digitado NÃO DEVE SER USADO.

Exemplo de execução:
Digite valores reais: 81 94 121 92 94 -10
Valor médio: 96.4
Quadrado da diferença: 148687.36

23) Teobaldo adora ver televisão. No entanto, sempre que ele muda de canal o volume do som deste novo canal fica diferente do anterior, às vezes com volume menor, outras vezes com volume maior. Então ele sempre precisa ajustar o volume pressionando uma quantidade de vezes seguidas os botões de aumentar e diminuir o volume. Além disso, a TV de Teobaldo possui um volume mínimo, com valor 0, e volume máximo, com valor 100; a TV nunca ultrapassa os volumes máximo e mínimo. Depois de tantas mudanças de volume, Teobaldo não sabe qual é o volume atual da TV. Escreva um programa em C++ que receba o volume inicial da TV, a quantidade de mudanças feitas e a quantidade de vezes que o botão foi apertado em cada mudança; e, ao final, mostre o volume atual da TV.

Exemplo de execução:
Volume inicial: 50
Quantidade de mudanças: 4
Valores das mudanças: 11 20 -15 -13
Volume atual: 53

Outro exemplo de execução:
Volume inicial: 50
Quantidade de mudanças: 5
Valores das mudanças: 30 30 30 40 -78
Volume atual: 22

24) Escreva um programa em C++ que deverá ler do teclado um conjunto de valores reais, onde os dois primeiros valores representam a largura e comprimento (em metros) de uma plataforma de terra existente em um supermercado, e os valores restantes (mínimo de 5 valores) representam uma cota (em centímetros) correspondente aos metros lineares desta plataforma. O final do conjunto de valores é indicado ao se digitar uma cota de valor nulo ou negativo (que NÃO DEVE SER CONSIDERADO nos cálculos solicitados). O proprietário quer fazer o asfaltamento dessa plataforma, aplainando-a através do acréscimo ou retirada de terra do terreno existente, de forma que a plataforma tenha ao final o valor da primeira cota informada originalmente. O programa deve imprimir o volume total (em m3) de terra retirada ou adicionada, conforme exemplos abaixo.

Exemplo de execução:

Largura e compr. da plataforma: 10 50
Digite as cotas da plataforma: 94 81 121 94 92 0
Volume de terra retirada: 60 m3

Outro Exemplo de execução:
Largura e compr. da plataforma: 10 50
Digite as cotas da plataforma: 120 94 121 92 81 0
Volume de terra adicionada: 460 m3

25) A sequência de Fibonacci é uma sequência de números que começa com 0, 1 e que daí por diante, cada elemento (e) terá um valor (v) igual à soma do último (u) como penúltimo (p) elemento, como demonstrado na tabela 1. Escreva um programa em

elemento	1	2	3	4	5	6	7	⋯
ultimo	-	-	1	1	2	3	5	⋯
penúltimo	-	-	0	1	1	2	3	⋯
valor (u + p)	0	1	1	2	3	5	8	⋯

Tabela 1: Valores da sequência de Fibonacci.

C++ que leia uma sequência de números, e que para cada número n lido, calcule a sequência dos n primeiros números de Fibonacci. Quando o usuário inserir um número negativo na entrada, o programa deverá parar a sua execução.

Exemplo de execução:
Entre com o valor: 1
Sequência: 0
Entre com o valor: 2
Sequência: 0 1
Entre com o valor: 3
Sequência: 0 1 1
Entre com o valor: 4
Sequência: 0 1 1 2
Entre com o valor: 11
Sequência: 0 1 1 2 3 5 8 13 21 34 55
Entre com o valor: -1
Fim de execução.

26) Escrever um programa em C++ que leia do teclado um par de números inteiros não negativos a e b, correspondentes aos limites inferior e superior, nesta ordem, de um intervalo [a,b], com a <= b. A seguir, ler ainda do teclado um número inteiro N(1 <= N <= 5) e um conjunto de N números inteiros (que denominaremos divisores). Na sequência, o programa deverá listar os números do intervalo lido, incluindo seus limites, que não são divisíveis por nenhum dos N números do conjunto de divisores lido. Ao final, mostrar a quantidade de números do intervalo que foram listados. Seu programa deve validar o intervalo informado, o valor de N e também a inexistência de zeros no conjunto de divisores informado.

Exemplo de execução: [a,b]: 25 15
Limites inválidos

Outro exemplo de execução:
[a,b]: 15 25
Qtde. divisores (N): 8
Qtde. Inválida

Outro exemplo de execução: [a,b]: 15 25
Qtde. divisores (N): 3
Divisor 1: 2
Divisor 2: 0
Inválido – reentre
Divisor 2: -3
Divisor 3: 4
Valores em [a,b] não divisíveis pelos divisores informados: 17 19 23 25
Números listados: 4

27) Um inteiro n é dito um número perfeito se ele é igual a soma de todos os seus divisores positivos, excluindo ele mesmo. Por exemplo, 6 é um número perfeito, pois seus divisores são 1, 2 e 3 e 1+2+3 = 6. O número 28 também é perfeito, pois 1+2+4+7+14=28. Faça um programa em C++ que leia uma sequência de números inteiros n1,n2,...,nk terminada por zero e informe ao usuário, para cada ni lido, se ele é um número perfeito ou não. O número 0 apenas indica o final da sequência e não deve ser considerado para a verificação da propriedade.

Exemplo de execução:
Entre com um inteiro: 2
2 não é um número perfeito.

Entre com um inteiro: 28
28 é um número perfeito.

Entre com um inteiro: 79
79 não é um número perfeito.

Entre com um inteiro: 8128
8128 é um número perfeito.
Entre com um inteiro: 0

28) Sua grana estava curta e então decidiu trabalhar temporariamente para o IBGE como agente do censo brasileiro. Sua remuneração depende de quantas residências são avaliadas por dia. Em cada residência, você faz o levantamento de quantas pessoas são homens e quantas são mulheres. Porém, o IBGE não pode pagar além de uma determinada quantidade de residências por dia. Escreva um programa em C++ que, após obter do usuário a quantidade de dias trabalhados e a cota máxima diária de residências que o IBGE pode pagar ao agente, obtenha para cada um dos dias trabalhados, pares de valores representando a quantidade de pessoas entrevistadas no levantamento de cada residência, sendo contabilizado primeiro os homens e depois as mulheres. O final dos dados de um dia é indicado quando o usuário fornece o par de valores (-1, -1). Ao final da execução, mostre a quantidade de casas avaliadas e de homens e mulheres entrevistados. Mostre também qual a média de homens e de mulheres por residência considerados na entrevista. OBS.: Residências avaliadas durante um mesmo dia que ultrapasse uma cota diária devem ser ignoradas nos cálculos de valores totais e médias.

Exemplo de execução:
Digite a quantidade de dias trabalhados: 3
Digite a cota diaria: 2
Dia 1:
1 2
3 4
5 6
-1 -1
Dia 2: 0 3
3 0
-1 -1
Dia 3:
5 1

-1 -1
Total de casas: 5
Total de homens: 12
Total de mulheres: 10
Media de homens por casa: 2.4
Media de mulheres por casa: 2

29) Escreva um programa em C++ que leia do teclado um conjunto de pares de números inteiros. O primeiro número de cada par poderá ser negativo, positivo ou nulo. O segundo número deverá ser diferente de zero e serve para indicar, de acordo com seu sinal, se uma série de números que deve ser gerada termina (caso de sinal negativo) ou inicia (sinal positivo) com o primeiro número informado. A série a ser gerada e exibida no monitor de vídeo deverá ser do tipo progressão aritmética, com razão e número de termos total dados pelo valor absoluto do segundo número digitado. Caso o segundo número do par informado seja nulo, deverá ser exibida mensagem ao usuário (vide exemplo) e desconsiderado o par. O final do conjunto de pares é assinalado com a informação de zeros para ambos os números do par, e este par não faz parte do conjunto. Ao final do processamento de cada par de números, o programa deverá mostrar a soma dos termos da série gerada e, ao final do programa, a quantidade de séries geradas (a partir de pares válidos).

Exemplo de execução:
Par de inteiros 1: -5 0
Segundo no. Inválido
Par de inteiros 1: 0 -5
Série gerada: -20 -15 -10 -5 0
Soma dos termos da série: -50
Par de inteiros 2: -20 10
Série gerada: -20 -10 0 10 20 30 40 50 60 70
Soma dos temos da série: 250
Par de inteiros 3: 100 7
Série gerada: 100 107 114 121 128 135 142
Soma dos termos da série: 847
Par de inteiros 4: 0 0
Detectado o final dos pares de números
Séries geradas: 3

30) Escreva um programa em C++ para auxiliar um viajante a escolher a melhor opção de abastecimento de seu veículo bicombustível (flex), se álcool ou gasolina. A

partir de médias de consumo na estrada analisadas pelo viajante, ele concluiu que seu carro abastecido com álcool tem um rendimento (em quilômetros rodados) de 78% a 82% do obtido com gasolina. Assim, se o preço que ele pagar pelo litro de álcool em relação ao litro de gasolina for inferior a esta faixa percentual, a melhor opção é abastecer com álcool. Acima desta faixa, a gasolina é mais econômica. Dentro da faixa (de 78% a 82%, inclusive) a opção de escolha é indiferente. O programa deverá receber do usuário os preços por litro do álcool e da gasolina e indicar a melhor opção de abastecimento. Para a escolha da melhor opção, deverá ser criada e usada a função escolha_comb(), que recebe como entradas os preços por litro do álcool e da gasolina e retorna um valor inteiro que indica a melhor opção: 1 (álcool); 2 (gasolina) ou 3 (indiferente). A função NÃO DEVE ler dados do teclado nem mostrar dados no monitor de vídeo.

Exemplo de execução:
Preço do litro de álcool: R$ 2.10
Preço do litro de gasolina: R$ 3.00
Sugestão de abastecimento: Álcool

Outro exemplo de execução do programa:
Preço do litro de álcool: R$ 2.35
Preço do litro de gasolina: R$ 2.70
Sugestão de abastecimento: Gasolina

Outro exemplo de execução do programa:
Preço do litro de álcool: R$ 2.40
Preço do litro de gasolina: R$ 3.00
Sugestão de abastecimento: Indiferente

31) Escreva um programa em C++ que obtenha do usuário 4 valores inteiros e imprima na tela mensagem indicando se o primeiro valor informado é divisor da soma dos outros 3 valores informados. Para testar esta relação entre os valores deverá ser criada e usada a função ehDivisor(), que recebe 4 valores inteiros como parâmetros, e retorna o valor 1 (um) se a relação se verifica, ou 0 (zero) caso contrário. A função NÃO DEVE ler dados do teclado nem mostrar dados no monitor de vídeo.

Exemplo de execução:
Indique 4 números inteiros: 3 5 8 2

3 é divisor.

Outro exemplo de execução do programa:
Indique 3 números inteiros: 2 3 4 8
2 não é divisor.

32) Escreva um programa que leia as coordenadas cartesianas (x,y) de 2 pontos, A e B no espaço e imprima na tela a menor distância entre os 2 pontos e também a distância do menor caminho de A até B passando por um ponto intermediário, C com coordenadas (0,0). Deve ser definida e usada a função de nome dist() que retorna a distância entre 2 pontos, cujas coordenadas são passadas como parâmetros. Esta função NÃO DEVE mostrar dados na tela ou solicitar valores do usuário. Obs.: Considere que a distância entre 2 pontos de coordenadas (xa,ya) e (xb,yb) é dada por distAB = $\sqrt{(xa-xb)^2+(ya-yb)^2}$. A distância do caminho de A até B passando por C = (0,0) é dado por caminho AB = distAC + distCB.

Exemplo de execução:
Digite coordenada de A: 10 50
Digite coordenada de B: 70 35
Menor distância entre A e B: 61.8466
Caminho de A a B, passando por (0,0): 129.253

33) Escreva um programa em C++ que obtenha do usuário um par de valores reais e um caracter indicando uma das 4 operações aritméticas: soma (+), subtração (−), multiplicação (x) e divisão (/). Ao final, o programa deve mostrar o resultado da operação indicada pelo usuário. Para fazer o cálculo solicitado, deve ser criada e usada a função calc(), que recebe 2 valores reais e um caracter como parâmetros e retorna o resultado do cálculo correspondente. Esta função NÃO DEVE mostrar dados na tela ou solicitar valores do usuário.

Exemplo de execução:
Operação: 4 10 x
Resultado: 40

34) Escreva um programa em C++ que solicita o total gasto pelo cliente de uma loja, e calcula o valor a vista com x% de desconto. O desconto é definido pelo usuário

(dono da loja). O programa usa uma função desc_a_vista(), que recebe do programa principal o valor gasto e o desconto dado pelo usuário e retorna o valor a ser pago com desconto. O programa então deve retornar o valor a ser pago. OBS.: A função NÃO DEVE mostrar dados na tela ou solicitar valores do usuário.

Exemplo de execução:
Total Gasto: 100
Desconto: 10
Total a pagar c/ desconto: 90

35) Escreva um programa em C++ que simule um jogo de “par ou ímpar” entre o usuário e o computador. O jogo deverá solicitar a aposta do usuário – 0 para apostar em par, 1 para apostar em ímpar – e um valor inteiro qualquer que ele jogará. A jogada do computador também será um inteiro qualquer, gerado pela função int rand(), que já existe no sistema, e que retorna um número aleatório qualquer. Ganha o jogo aquele que adivinhar qual o tipo da soma dos valores apresentados pelo computador e pelo usuário, se par ou impar, conforme a aposta de cada um. O seu programa deverá chamar a função par_ou_impar(), de sua autoria, que receba um inteiro indicando a aposta do usuário (par ou ímpar), um inteiro indicando o valor que o usuário jogou, e que retorne 1 caso o usuário tenha vencido ou 0 caso o computador tenha vencido. OBS.: A função NÃO DEVE mostrar dados na tela ou solicitar valores do usuário.

Exemplo de execução:
Entre com a sua aposta (0:Par, 1:Ímpar) e o valor da sua jogada: 0 4
O computador ganhou

Outro exemplo de execução:
Entre com a sua aposta (0:Par, 1:Ímpar) e o valor da sua jogada: 1 33
Parabéns, você venceu!

36) Escreva um programa em C++ para calcular e mostrar a distância de um ponto (x,y) do plano cartesiano à origem dos eixos (0,0). A distância procurada é obtida pela raiz quadrada da soma dos quadrados dos valores das coordenadas x e y de cada ponto. Para resolver o problema proposto, pede-se o desenvolvimento e uso de uma função de nome distxy(), que recebe como entradas as coordenadas (x,y)

de um ponto, calcula e devolve ao programa que a chamar a distância procurada. OBS.: A função NÃO DEVE mostrar dados na tela ou solicitar valores do usuário.

Exemplo de execução:
(x,y): 3 4
Distância calculada: 5

Outra execução:
(x,y): 0 -8
Distância calculada: 8

Outra execução:
(x,y): -1 -1
Distância calculada: 1.414

37) Sabe-se que 4 segmentos podem formar um quadrilátero apenas se a medida do maior segmento for menor que a soma das medidas dos outros 3. Escreva um programa em C++ que recebe do usuário as medidas de 4 segmentos, verifica sua validade (valores reais positivos e não nulos), avalia a possibilidade de formar algum quadrilátero com as medidas informadas e, caso possível, calcula e mostra o perímetro do polígono obtido. Para a verificação da possibilidade de formar o quadrilátero, deve ser definida e usada a função analisa4(), que recebe como entrada as medidas de 4 segmentos e devolve ao programa que a chamou: -1, caso alguma das medidas informadas seja inválida; 0, caso as medidas sejam válidas mas não permitam a formação de um quadrilátero; OU o perímetro do quadrilátero, caso as medidas sejam válidas e possibilitem a formação do mesmo. OBS.: A função NÃO DEVE mostrar dados na tela ou solicitar valores do usuário.

Exemplo de execução:
Medidas (4 segmentos): 10 2 -5 6
Medidas inválidas

Outro exemplo de execução:
Medidas (4 segmentos): 10 2 0.5 6
Medidas não formam um quadrilátero

Outro exemplo de execução:
Medidas (4 segmentos): 10 2 5 6
Perímetro do quadrilátero formado: 23

38) Uma loja tem um plano de desconto progressivo para os seus clientes, onde quanto mais eles compram, mais desconto eles ganham. Além disso, a loja tem um plano de fidelidade que garante um desconto alternativo ainda maior aos clientes mais assíduos, de acordo com a tabela abaixo:

Valor da compra	Desconto Normal	Desconto Fidelidade
Até R$ 200,00	0%	2%
de 200,01 a 300,00	5%	7%
de 300,01 a 550,00	10%	13%
A partir de 550,01	15%	19%

Escreva um programa em C++ que peça ao usuário o valor de uma compra e o tipo do cliente que a efetuou. EM seguida o programa deve exibir o valor dado como desconto e o valor que foi efetivamente pago pelo cliente. Para calcular o desconto, deve ser definida e usada a função chamada calc_desc(), que recebe o valor total da compra do cliente e um valor que indique se o cliente tem plano de fidelidade (0 se não tiver, 1 se tiver) e que retorne o valor em reais do desconto aplicado ao valor da compra. OBS.: A função NÃO DEVE mostrar dados na tela ou solicitar valores do usuário.

Exemplo de execução:
Valor da compra e tipo de cliente: 1072.30 1
Total de desconto: 203.73
Total a pagar: 868.56

Outro exemplo de execução:
Valor da compra e tipo de cliente: 247.37 0
Total de desconto: 12.36
Total a pagar: 235.01

39) Faça um programa em C++ que verifique se a quantidade em estoque de um produto de uma loja acabou ou não toda vez que houve uma retirada. O programa deve pedir ao usuário a quantidade inicial do produto, e depois solicitar do usuário a quantidade vendida do produto, informando quando não houver quantidade suficiente em estoque para a venda. O programa termina mostrando a quantidade final do estoque. O programa deve usar a função estoque() para verificar se há

quantidade suficiente e para atualizar a quantidade do estoque. Ela deve receber como parâmetros a quantidade em estoque e a quantidade vendida, e deve devolver o novo valor do estoque, que deve permanecer o mesmo caso o estoque não consiga cobrir a quantidade vendida. OBS.: A função NÃO DEVE mostrar dados na tela ou solicitar valores do usuário.

Exemplo de execução:
Qtde. inicial do estoque: 20
Qtde. Vendida: 12
Qtde. em estoque: 8

Outro exemplo execução:
Qtde. inicial do estoque: 13
Qtde. vendida: 20
Estoque insuficiente: 13

40) Faça um programa em C++ que receba do usuário a medida do raio de uma circunferência e uma escolha: 1 se o usuário deseja saber o perímetro da circunferência, 2 se o usuário deseja saber a área do círculo correspondente. O programa deverá usar uma função de nome calcCirculo() que recebe o raio e a escolha e retorna o valor do cálculo realizado. A entrada e saída de dados deve ser completamente manipulada pelo programa principal. OBS.: A função NÃO DEVE mostrar dados na tela ou solicitar valores do usuário.

Exemplo de execução:
Informe raio da circunf.: 3
Deseja perimetro (1) ou área (2)?: 1
Perímetro = 18.8496

Outro exemplo de execução:
Informe raio da circunf.: 3
Deseja perimetro (1) ou área (2)?: 2
Área = 28.2743

41) Escreva um programa em C++ que obtenha do usuário 4 valores inteiros e imprima na tela mensagem indicando se a soma dos dois maiores números é divisível pelo ou divisor do produto dos 3 menores valores informados. Para testar esta relação entre os valores deverá ser criada e usada a função ehDivisor(), que

recebe 4 valores inteiros como parâmetros, e retorna o valor 1 (um) se a relação “soma divisível pelo produto” se verifica, 2 (dois) se a relação “soma divisor do produto” se verifica, ou 0 (zero) caso contrário. OBS.: A função NÃO DEVE mostrar dados na tela ou solicitar valores do usuário.

Exemplo de execução:
Indique 4 números inteiros: 1 7 3 2
Propriedade não se verifica.

Outro exemplo de execução do programa:
Indique 4 números inteiros: 2 3 4 8
12 é divisor de 24.

Outro exemplo de execução do programa:
Indique 4 números inteiros: 20 1 2 4
24 é divisível por 8.

42) Depois de passar muito tempo programando e esquecendo-se de se exercitar, Teobaldo engordou mais do que deveria e decidiu começar a fazer atividades físicas para queimar calorias. Ele deseja construir um programa em C++ que monitore seus treinos informando quantos quilos ele perdeu por dia até que ele alcance um determinado peso desejado. Este programa deve receber de Teobaldo seu peso inicial e sua meta final de peso, e em seguida receba dele diversos valores de calorias gastas em cada treino, mostrando na tela o peso após o treino e se ele teve ou não um bom rendimento. O programa deve solicitar novos valores de calorias até que o peso atual de Teobaldo seja menor que o peso desejado por ele, considerando que cada valor informado representa o resultado de um novo treino e, consequente, um novo gasto de calorias. Para calcular o rendimento e o peso após o treino, deve ser definida e usada a função de nome emagrece() que recebe: (i) o peso atual de Teobaldo (emquilos) e (ii) a quantidade de calorias que ele gastou no dia e retorna: (a) o novo peso de Teobaldo e (b) 1 ou 0, caso Teobaldo tenha feito um treino bom ou ruim, respectivamente. Considere que a cada 7000 calorias gastas, Teobaldo emagrece 1 quilo; e, um treino bom é entendido como um treino onde o novo peso de Teobaldo é menor que 99% de seu peso anterior.

Exemplo de execução:
Informe seu peso: 85

Informe sua meta: 82
Informe calorias perdidas: 7000
Seu novo peso é: 84
Você teve um bom rendimento!
Informe calorias perdidas: 5000
Seu novo peso é: 83.2857
Você precisa treinar mais...
Informe calorias perdidas: 10000
Seu novo peso é: 81.8571
Você teve um bom rendimento!
Você atingiu sua meta!

43) Teobaldo estuda o clima e está realizando um experimento de análise de temperaturas. Após coletar um conjunto de temperaturas em Grau Celsius (ºC), Teobaldo precisa converter cada um dos valores para Grau Fahrenheit (◦F) e comparar se cada valor está acima da média correspondente. Para ajudar Teobaldo, sua tarefa é escrever um programa em C++ que receba do usuário um conjunto de N pares de valores, onde cada par consiste de uma temperatura em Grau Celsius e de um valor médio em Grau Fahrenheit. Para cada valor em Grau Celsius, seu programa deve mostrar na tela o valor convertido para Grau Fahrenheit e se está acima da média. Ao final, seu programa deve mostrar na tela o total de valores acima da média. Para auxiliar nesta tarefa, você deve definir e utilizar a função converte_temp(), que recebe dois parâmetros de entrada - a temperatura em Celsius e a média em Fahrenheit, e devolve dois outros valores - a temperatura Celsius convertida para Grau Fahrenheit e um número inteiro (1 - se a temperatura convertida está acima da média, 0 - caso contrário). Lembre-se: ◦F = ◦C ×1,8 + 32.

Exemplo de execução:
Quantidade de Temperaturas: 3
Tempertura (C) e média (F): 28 80
28 C = 82.4 F --- Está acima da média
Tempertura (C) e média (F): 30 95
30 C = 86 F --- Está abaixo da média
Tempertura (C) e média (F): 25 70
25 C = 77 F --- Está acima da média
Existem 2 temperaturas acima da média

44) Escreva um programa em C++ que recebe via teclado um valor N e mostre a quantidade de mudanças de sinal no valor y e o primeiro valor de x onde houve

mudança de sinal de y para um conjunto de N pares (x,y) de valores reais fornecidos pelo usuário. Para determinar estes valores deve ser definida e usada a função de nome raizes(), que recebe como um parâmetro o número N, solicite do usuário o fornecimento de N depares(x,y) de valores reais e devolva quantidade de mudanças de sinal no valor y e o primeiro valor de x onde houve mudança de sinal de y. A função raizes() NÃO DEVE mostrar os resultados calculados na tela, mas note que esta função solicita do usuário APENAS os valores dos N pares (x,y) de valores reais.

Exemplo de execução:
N: 8
2 -15
3 -8
4 -2
5 3
6 7
7 1
8 -4
9 -7
Mudanças de sinal: 2
1o. x com mudança de sinal de y: 5

45) Escreva um programa em linguagem C++ que recebe via teclado um valor N com a quantidade de Produtos a serem informados e um conjunto de dados de Produto, contendo o Tipo do Produto (inteiro), seu Estoque atual (real), e seu Preço (real). Para cada conjunto digitado o programa deverá informar, se necessário, a quantidade necessária para repor o estoque mínimo. No final do processamento o programa deverá informar a quantidade total de itens listados e a quantidade de itens com estoque abaixo do mínimo. Deverá listar o valor total do estoque (somatório dos Preço x Quantidade em Estoque, de todos os produtos) e o valor total pendente de reposição (somatório dos Preço x Quantidade de Reposição, de todos os produtos com estoque abaixo do mínimo). Para o verificar se um produto existe no estoque, deve ser defina e usada a função calc_reposicao(), que recebe como parâmetros o Tipo (inteiro) e Quantidade (real), retornando um valor 1 (um) se a quantidade está abaixo do estoque mínimo, ou 0 (zero) caso contrário. A função deve também devolver um valor real com a quantidade necessária para atingir o

estoque mínimo do Produto. Os níveis de estoque mínimo são dados pela tabela abaixo:

Tipo	Estoque Mínimo
100	40
200	20
300	10
400	5

Exemplo de execução:
Quantidade de produtos: 5
Em cada produto, indique tipo,estoque,preço
Produto 1: 200 18 5.00
Repor o estoque em 2 unidades !!
Produto 2: 300 10 6.00
Produto 3: 100 35 4.00
Repor o estoque em 5 unidades !!
Produto 4: 100 40 2.00
Produto 5 400 10 3.00
5 prod. no estoque, valor total = 400 reais.
2 prod. abaixo do minimo, com valor pendente = 30 reais.

46) Hermenegildo é um passional investidor da bolsa de valores que, cansado de perder dinheiro, resolveu estipular que a partir de agora: (i) Se ele estiver num lucro de 5% numa operação, ele deverá vender metade das ações para garantir algum dinheiro no bolso e se proteger, caso o valor das ações caia;(ii) Se ele estiver num lucro de 10% numa operação, ele deverá vender todas as ações e tirar o dia de folga; (iii) Se ele estiver num prejuízo de 1,5% numa operação, ele deverá vender todas as ações e ir se benzer; (iv) Nos demais casos ele deverá permanecer com as ações compradas. Escreva um programa em C++ para Hermenegildo que funcione da seguinte forma: o programa lerá um número n, e em seguida n pares de números, cada par representando o valor de compra de uma ação e o valor atual da ação. Para cada par de valores, o programa deverá calcular qual é o lucro ou o prejuízo da operação, e informar o que ele deve fazer em cada caso. Para isso, o programa utilizará a função segura_ou_vende(), que recebe como parâmetro o valor de compra da ação, o valor atual da ação e um indicador de situação. A função deverá fazer os cálculos e devolver o valor percentual de lucro ou prejuízo no

indicador de situação. Além disso, a função deve retornar: (i)-1 se ele tiver que vender todas as ações no prejuízo; (ii) 0 se ele não tiver de fazer nada; (iii) 1 se ele tiver que vender metade das ações no lucro; (iv) 2 se ele tiver que vender todas as ações no lucro. OBS.: A função NÃO DEVE mostrar dados na tela ou solicitar valores do usuário.

Exemplo de execução:
Número de operações: 4
Valores de compra e atual, dois a dois:
19.43 21.50
33.17 34.70
25.19 24.93
21.87 21.44
Resultado:
Operaçao 1: 10.6%, venda tudo e tire folga
Operaçao 2: 4.6%, aguarde a definição
Operaçao 3: -1%, aguarde a definição
Operaçao 4: -1.9%, venda tudo e se benza !!

47) Dizem no Brasil que um professor ganha mal e que seu salário não dá pra nada. Por isso, a OPB (Ordem dos Professores do Brasil) contratou você para ajudá-los a melhorar a vida dos professores brasileiros. Faça um programa completo em C++ que verifique se houve alteração significativa de preço dos produtos que os professores compram no mercado. Crie uma função que verifique se o preço de um produto foi alterado nesta semana comparado à semana passada. Se houve um aumento de 10% ou mais, retorne valor 2; se houve uma queda de 10% ou mais, retorne valor 1; para outros casos, retorne valor 0. Adicionalmente, calcule qual foi a percentagem de aumento ou de redução de preços atualizando um índice de percentagem (também passado à função). Seu programa deve ler um valor n relativo a quantos produtos serão verificados. Para cada produto, deve-se obter um par de valores, referente ao preço da semana passada e da semana atual, respectivamente. Informe ao professor o que houve com cada um dos produtos nessa semana em relação à semana anterior e de quanto foi o aumento ou a redução (em percentagem), quando existir alteração significativa nos preços. OBS.: A função NÃO DEVE mostrar dados na tela ou solicitar valores do usuário.

Exemplo de execução:

Quantos produtos? 3
Preco do produto (passado e atual)? 3 4
Aumento de 33.3333% no preco do produto
Preco do produto (passado e atual)? 10 9.5
Nao houve mudanca significativa
Preco do produto (passado e atual)? 1 0.1
Houve reducao de 90% no preco do produto

48) Teobaldo é um daqueles eleitores que insistem em acreditar que a eleição pode ser anulada se mais de 50% dos votos forem nulos. Como este ano nenhum dos candidatos lhe agrada, ele fará seu “voto de protesto”. Contudo, gostaria de saber com antecedência se seu “protesto” não será em vão, pedindo sua ajuda para montar um programa que será utilizado para uma pesquisa eleitoral online. Faça um programa em C++ que leia um número inteiro n indicando o total de eleitores e, em seguida, leia n números inteiros representando os votos dos eleitores, codificados da seguinte maneira: 42 para o candidato A, 18 para o candidato B e 35 para o candidato C. Qualquer voto diferente desses valores deve ser considerado nulo. Faça uma função chamada voto() que recebe 4 parâmetros, sendo os 3 primeiros as quantidades de votos dos candidatos A, B e C, respectivamente, e o último parâmetro o número a ser votado por um eleitor. Essa função deve atualizar a contagem dos votos, se o número votado for válido, e retornar 0, ou simplesmente retornar 1, caso o voto seja nulo. Ao final, seu programa deve indicar se a eleição será anulada segundo Teobaldo ou qual candidato está na frente (se dois ou mais estiverem empatados em primeiro lugar, mostre qualquer um deles). Você deve utilizar a função voto() no seu programa principal. OBS.: A função NÃO DEVE mostrar dados na tela ou solicitar valores do usuário.

Exemplo de execução:
7
42
01
42
18
29
99
35
O candidato A está na frente.

Outro exemplo execução:
5
99
01
42
67
35
A eleição será anulada segundo Teobaldo.

49) Codifique um programa em C++ para auxiliar um leiloeiro a identificar os três maiores lances em suas atividades de vendas. O programa deverá receber vários lances para compra de um produto em leilão, onde cada lance é caracterizado por um código do comprador (inteiro, positivo e diferente de 0), e por um valor do lance, em reais. Quando um lance possuir 0 no código do comprador ou no valor da oferta, significa que o leilão acabou e o programa deverá mostrar os três maiores lances, em ordem decrescente, e encerrar o processamento. Para identificar os três maiores lances defina e utilize a função Seleciona_Melhores_Ofertas() que recebe cada lance (com código do comprador e valor do lance) juntamente com as variáveis para armazenar os três melhores lances, cada um composto do código do comprador e do valor de seu lance. Caso o valor do lance seja menor ou igual ao maior valor de lance existente, a função deverá retornar 0 (significando que o lance é inválido), e em caso contrário deverá retornar +1. Caso a função retorneovalor0, o programa deverá dar mensagem de “Lance inválido: Ignorado.” , e em caso contrário deverá dar mensagem de “Lance OK: Processado.” . OBS.: A função NÃO DEVE mostrar dados na tela ou solicitar valores do usuário.

Exemplo de execução:
Digite o Lance 1: 4 1012.50
Lance OK: Processado.
Digite o Lance 2: 1 980
Lance inválido: Ignorado.
Digite o Lance 2: 3 1050
Lance OK: Processado.
Digite o Lance 3: 4 1200.99
Lance OK: Processado.
Digite o Lance 4: 5 1250
Lance OK: Processado.
Digite o Lance 5: 3 1300
Lance OK: Processado.

Digite o Lance 6: 0 0
Foram processados 5 Lances.
1o. melhor Lance: 1300 do comprador 3
2o. melhor Lance: 1250 do comprador 5
3o. melhor Lance: 1200.99 do comprador 4

Outro exemplo de execução:
Digite o Lance 1: 4 1012.50
Lance OK: Processado.
Digite o Lance 2: 3 1050
Lance OK: Processado.
Digite o Lance 3: 0 0
Foram processados 2 Lances.
1o. melhor Lance: 1050 do comprador 3
2o. melhor Lance: 1012.50 do comprador 4

50) Fulano, Beltrano e Cicrano estão disputando quem tem mais sorte. Para isso, eles pegaram um dado de seis lados e, um de cada vez, lançou o dado e anotou o número obtido. Eles fizeram diversas jogadas e agora querem descobrir quem foi o vencedor de cada rodada. Faça um programa completo em C++ que os auxilie a descobrir quantas rodadas cada um venceu. Obtenha do teclado o número de rodadas que foram realizadas e crie uma função chamada vencedor_rodada() que recebe como parâmetros os números que cada um tirou em uma determinada rodada. Faça sua função retornar 1, 2 ou 3 caso Fulano, Beltrano ou Cicrano tenha vencido aquela rodada, nessa ordem. Ao final do seu programa, mostre quantas rodadas cada jogador venceu. Obs. 1: Assuma que houve apenas um vencedor, sem empates durante as rodadas. OBS. 2: A função NÃO DEVE mostrar dados na tela ou solicitar valores do usuário.

Exemplo de execução:
Digite numero de rodadas do jogo: 3
Digite numeros obtidos pelos jogadores:
1 2 3
4 5 6
1 3 6
Fulano venceu 0 vezes
Beltrano venceu 0 vezes
Cicrano venceu 3 vezes

Outro exemplo de execução:
Digite numero de rodadas do jogo: 3

Digite numeros obtidos pelos jogadores:
6 2 1
1 2 4
4 1 5
Fulano venceu 1 vezes
Beltrano venceu 0 vezes
Cicrano venceu 2 vezes

51) O pôquer curitibano é uma variante do jogo na qual cada jogador começa com duas cartas e, a cada rodada, pode escolher trocar uma de suas cartas por uma carta comprada do baralho. Nessa variante, apenas as cartas 2,3,⋯ ,9,10 e A (Ás) são utilizadas, sendo que a pontuação de cada carta é igual ao seu valor, exceto a carta A que tem pontuação 14. Ganha o jogador que, após k rodadas, tenha na mão o par de maior valor ou as cartas de maiores valores (caso nenhum jogador tenha um par). Faça um programa em C++ que auxilie um jogador a tomar as decisões mais lógicas numa partida de pôquer curitibano. Crie uma função chamada troca_carta() que recebe três parâmetros: as duas cartas que o jogador tem em mãos e a carta que foi comprada do monte nessa rodada. Essa função deve verificar se é vantajoso trocar alguma carta da mão pela comprada e retornar as cartas trocadas, se for o caso, e um inteiro representando se houve troca (1) ou não (0). No programa principal, leia o número k de rodadas da partida e dois inteiros representando as cartas que o jogador tem na mão no início do jogo (14 representa a carta A). Para cada uma das k rodadas, leia um inteiro representando a carta comprada. Utilizando a função troca_carta() seu programa deve obter, em cada rodada, uma mão melhor que a anterior, se possível. Caso haja troca de cartas a mensagem “apostar!” deve ser impressa. Caso contrário, a mensagem “mesa.” é que deve ser mostrada. Se o jogador terminar a última rodada com um par de áses na mão, a mensagem “apostar tudo!!!” deve ser impressa, independente da rodada na qual o par foi obtido. OBS.: A função NÃO DEVE mostrar dados na tela ou solicitar valores do usuário.

Exemplo de execução:
Número de rodadas: 3
Mão inicial: 2 7
Rodada 1: 3
Mão: 3 7 -- apostar!

Rodada 2: 3
Mão: 3 3 -- apostar!
Rodada 3: 14
Mão: 3 3 – mesa.

Outro exemplo de execução:
Número de rodadas: 3
Mão inicial: 14 3
Rodada 1: 2
Mão: A 3 -- mesa.
Rodada 2: 14
Mão: A A -- apostar!
Rodada 3: 10
Mão: A A -- apostar tudo!!!

52) Daenerys e Drogo entendiados inventaram um jogo de adivinhação. Neste jogo o oponente 1 inventa uma senha que consiste de um número de 3 dígitos não repetidos entre 1 e 4. São exemplos de senhas VÁLIDAS: 123, 231 e 412; As senhas 212 e 315 são senhas INVÁLIDAS. O oponente 2 deverá dar palpites para adivinhar a senha. Para cada palpite feito devem ser dadas as seguintes dicas: (a) quantos dígitos do palpite aparecem em qualquer posição na senha; (b) quantos dígitos do palpite aparecem na posição correta na senha (vide Exemplo de execução abaixo). Sua missão é implementar este jogo no computador. Para isso você deve escrever uma função em C++de nome dicas() que recebe como parâmetros uma senha e um palpite. A função deve devolver a quantidade de dígitos do palpite que aparecem em senha (acertdig) e a quantidade de dígitos do palpite que aparecem em senha na mesma posição (acertdigpos). Além disso, a função deve retornar o valor 0 (zero) se o palpite for incorreto, ou 1 (um) se o palpite estiver correto. Escreva também o programa principal que lê a senha do oponente 1 e depois lê os palpites do oponente 2 e, usando os resultados da função dicas(), imprime as dicas correspondentes até que este acerte a senha. O programa somente termina quando o oponente 2 adivinha a senha. OBS.: A função NÃO DEVE mostrar dados na tela ou solicitar valores do usuário.

Exemplo de execução:
Oponente 1, entre sua senha: 341
Oponente 2, adivinhe a senha: 324
DICAS

Qtd. dígitos corretos: 2
Qtd. dígitos corretos na pos. correta: 1
Oponente 2, adivinhe a senha: 321
DICAS
Qtd. dígitos corretos: 2
Qtd. dígitos corretos na pos. correta: 2
Oponente 2, adivinhe a senha: 421
DICAS
Qtd. dígitos corretos: 2
Qtd. dígitos corretos na pos. correta: 1
Oponente 2, adivinhe a senha: 341
Você acertou. Parabéns!

53) Faça um programa em C++ para realizar o agendamento de consultas ambulatoriais do SUS. Dois tipos de consultas são realizadas: (1) consulta normal e (2) consulta de retorno. Uma consulta normal possui 30 minutos de duração, enquanto uma consulta de retorno requer 15 minutos. Primeiro, o usuário deve informar a carga horária do médico em minutos. Depois, os agendamentos são realizados informando o tipo da consulta para cada paciente enquanto há carga horária suficiente para mais um agendamento. Ao final, o programa deve informar o número total de consultas agendadas. Para resolver o problema proposto pede-se o desenvolvimento e uso de uma função de nome agendamento() que recebe como entrada o tipo de consulta e a carga horária disponível, e devolve esta carga horária atualizada de acordo com o tempo da consulta, devolvendo também o valor 1 (um) caso a consulta tenha sido agendada ou 0 (zero) caso contrário. OBS.: A função NÃO DEVE mostrar dados na tela ou solicitar valores do usuário.

Exemplo de execução:
Carga horaria: 90
Informe agendamentos:
1
Consulta agendada.
1
Consulta agendada.
2
Consulta agendada.
1
Consulta nao agendada.
2
Consulta agendada.
Total de consultas agendadas: 4

Outro exemplo de execução:
Carga horaria: 50
Informe agendamentos:
2
Consulta agendada.
1 Consulta agendada.
Total de consultas agendadas: 2

54) Escreva um programa em C++ para ler um vetor de inteiros de tamanho N (N estabelecido via diretiva #define) e um valor X. O programa deve exibir na tela ACHEI se o valor X existir pelo menos uma vez no vetor e NÃO ACHEI caso contrário. O programa deve também alterar o vetor dividindo todos os seus valores por X. Ao final, o programa deve exibir na tela o conteúdo do vetor. OBS.: Para a exibição do conteúdo do vetor na tela, considere que já existe a função void mostra_vetor (int vet[]). Sua solução deve apenas chamar esta função.

Exemplo de execução: (com N=6)
Informe vetor: 4 3 12 6 9 3
Informe X: 3
ACHEI.
Vetor final: 1.33 1 4 2 3 1

55) Escreva um programa em C++ que leia um vetor de números inteiros de tamanho N (N estabelecido via diretiva #define). O programa deve calcular quantas vezes ocorrem valores repetidos em posições consecutivas ("vizinhas") no vetor. Além disso, o programa deve atribuir valor 0 nestas posições do vetor. Ao final, o programa deve exibir na tela o conteúdo do vetor. OBS.: Para a exibição do conteúdo do vetor na tela, considere que já existe a função void imprime_vetor (int vet[]). Sua solução deve apenas chamar esta função.

Exemplo de execução (com N=10):
Informe vetor: 2 3 3 4 5 1 5 5 7 8
Qtde. de repetições: 2
Vetor final: 2 0 0 4 5 1 0 0 7 8

Outro exemplo de execução (com N=7):
Informe vetor: 2 3 3 3 2 1 5
Qtde. de repetições: 2

Vetor final: 2 0 0 0 2 1 5

56) Escreva um programa em C++ que leia um vetor de inteiros de tamanho N (N estabelecido via diretiva #define) e um valor X. O programa deve calcular o valor que deve ser somado a cada elemento do vetor para que este atinja o valor X. Este resultado deve substituir o valor correspondente no vetor. Além disso, deve ser calculada e exibida na tela a média aritmética dos valores finais do vetor. Ao final, o programa deve exibir na tela o conteúdo do vetor já alterado. OBS.: Para a exibição do conteúdo do vetor na tela, considere que já existe a função void mostra_vetor (int vet[]). Sua solução deve apenas chamar esta função.

Exemplo de execução: (com N=6)
Informe vetor: 4 3 12 6 9 3
Informe X: 3
Média: -3
Vetor final: -1 0 -9 -2 -6 0

Outro exemplo de execução: (com N=6)
Informe vetor: 42 23 21 36 79 53
Informe X: 50
Média: 7.666
Vetor final: 8 27 29 14 -29 -3

57) Faça um programa em C++ que permita consultar a classificação de equipes no mundial de construtores de automobilismo. Cada equipe possui dois pilotos: um piloto principal e um piloto secundário. Considere dois vetores para representar esses pilotos: (i) vetor com a pontuação dos pilotos principais de cada equipe e (ii) vetor com a pontuação dos pilotos secundários de cada equipe. Considere também que pilotos da mesma equipe estão em posições opostas em cada vetor. Por exemplo: o primeiro piloto principal e o último piloto secundário são da mesma equipe, o segundo piloto principal e o penúltimo piloto secundário são de outra equipe, e assim por diante. A pontuação dos pilotos não está necessariamente ordenada. O número N de pares de pilotos que disputam o campeonato é fixo e deve ser informado via #define. Seu programa deve obter a pontuação de cada piloto principal, de cada piloto secundário e calcular a pontuação de cada equipe. Guarde a pontuação de sua respectiva equipe no lugar da pontuação original de

cada piloto. Mostre como está a classificação das equipes, imprimindo as posições dos dois vetores. Mostre também quem é a equipe que lidera o campeonato. OBS.: Para a exibição do conteúdo do vetor na tela, considere que já existe a função void imprime_vetor (int vet[]). Sua solução deve apenas chamar esta função.

Exemplo de execução (para N=3):
Pontos p/ piloto principal: 5 2 4
Pontos p/ piloto secundário: 3 6 9
Pontuacao por equipes: 14 8 7 7 8 14
Equipe 1 na lideranca

Outro exemplo de execução (para N=5):
Pontos p/ piloto principal: 5 2 4 6 3
Pontos p/ piloto secundario: 1 9 0 10 8
Pontuacao por equipes: 13 12 4 15 4 4 15 4 12 13
Equipe 4 na lideranca

58) Faça um programa em C++ que leia (do teclado) um vetor de tamanho N (N estabelecido via diretiva #define) de números reais e crie um novo vetor, de mesmo tamanho e tipo do original, com as seguintes características: (a) valores dos elementos das extremidades do novo vetor (primeiro e último) idênticos aos do original; (b) valores dos elementos do “miolo” do vetor (segundo ao penúltimo) obtidos a partir da média dos valores de seu antecessor e sucessor posicionais no vetor original, desde que seu valor esteja situado acima do valor de seu antecessor e abaixo do valor de seu sucessor no vetor original; não verificada esta característica, gerar o elemento do vetor de saída com o mesmo valor do original. Ao final do processamento, o programa deverá mostrar o vetor gerado no monitor de vídeo e informar quantas substituições de valores foram feitas. OBS.: Considere que a função void imprime_vetor(float vet[]) para mostrar o vetor criado JÁ EXISTE. Sua solução deverá apenas CHAMAR esta função onde necessário.

Exemplo de execução (com N=10):
Informe vetor: 3 4.5 2 5 6 7.3 7.5 9 12 15
Vetor gerado: 3 4.5 2 4 6.15 6.75 8.15 9.75 12 15
Alterações: 6

59) Faça um programa em C++ que leia um vetor A de 6 elementos contendo o gabarito da Mega Sena. A seguir, ler um vetor B de 10 elementos contendo uma aposta. Escrever quantos pontos fez o apostador, e se ele fez a sena (6 acertos), a quina (5 acertos) ou a quadra (4 acertos). E no vetor B, mudarovalordoelementoqueteveacertopara99. OBS.: Considere que a função void imprime_vetor(int vet[]) para mostrar o vetor criado JÁ EXISTE. Sua solução deverá apenas CHAMAR esta função onde necessário.

Exemplo de execução:
Vetor A: 7 43 51 47 6 3
Vetor B: 7 43 9 1 5 8 22 47 6 34
Fez a quadra 99 99 9 1 5 8 22 99 99 34

60) A sequência de Fibonacci é uma sequência de números que começa com 0, 1 e que daí por diante, cada elemento (e) terá um valor (v) igual à soma do último (u) com o penúltimo (p) elemento, como demonstrado na tabela1. Escreva um programa em

elemento	1	2	3	4	5	6	7	⋯
último	-	-	1	1	2	3	5	...
penúltimo	-	-	0	1	1	2	3	⋯
valor (u + p)	0	1	1	2	3	5	8	⋯

Tabela 1: Valores da sequência de Fibonacci.

C++ que leia uma sequência de números, e que para cada número n lido, calcule a sequência dos n primeiros números de Fibonacci. Quando o usuário inserir um número negativo na entrada, o programa deverá parar a sua execução.

Exemplo de execução:
Entre com o valor: 1
Sequência: 0
Entre com o valor: 2
Sequência: 0 1
Entre com o valor: 3
Sequência: 0 1 1
Entre com o valor: 4
Sequência: 0 1 1 2

Entre com o valor: 11
Sequência: 0 1 1 2 3 5 8 13 21 34 55
Entre com o valor: -1
Fim de execução.

61) Considere um programa que gerencia uma prateleira de locadora de filmes. A prateleira é representada no programa por um vetor, onde cada posição equivale a um filme e à quantidade de exemplares disponíveis para a locação. Por exemplo, o número 3 na posição 7 indica que existem 3 exemplares disponíveis do filme que está na posição 7. O valor 0 indica que não há exemplar disponível para ser alugado onde quer que ele esteja. Escreva um programa em C++ que leia N quantidades de exemplares (N estabelecido via #define), e que depois de efetuar a leitura da situação atual da prateleira, gerencie a locação e devolução dos filmes da seguinte forma: o usuário deverá digitar f para finalizar, a para indicar que será uma locação, e d para indicar uma devolução. Após selecionar a ou d, o usuário deverá digitar uma sequência de tuplas representando a posição do filme na prateleira e a quantidade de filmes a serem locados ou devolvidos. O valor -1 encerra esta leitura. A cada etapa de aluguel ou devolução, o programa deverá mostrar a situação da prateleira até que o usuário digite a opção f para finalizar a execução. Se não houver quantidade suficiente para a venda, esta não deve ser feita e a prateleira fica sem mudanças. OBS.: Considere que a função void imprime_vetor(int vet[]) para mostrar o conteúdo de um vetor JÁ EXISTE. Sua solução deverá apenas CHAMAR esta função onde necessário.

Exemplo de execução (com N=5):
Entre situação da prateleira: 2 4 0 1 2
(a)lugar? (d)evolver ou (f)inalizar? a 0 1
Alugado filme 0.
2 1
Filme 2 sem exemplares.
3 1
Alugado filme 3. -1 -1
Situação da prateleira: 1 4 0 0 2
(a)lugar? (d)evolver ou (f)inalizar? d 2 2
Devolvido filme 2. 3 1
Devolvido filme 3. -1 -1
Situação da prateleira: 1 4 2 1 2
(a)lugar? (d)evolver ou (f)inalizar? a 4 4

Exemplares do filme 4 insuficientes.
-1 -1
Situação da prateleira: 1 4 2 1 2
(a)lugar? (d)evolver ou (f)inalizar? f
Fim da execução.

62) Escreva um programa em C++ para ler um vetor de N elementos inteiros (N estabelecido via #define). Em seguida, pergunte ao usuário qual posição deve ser eliminada. A partir dessa posição o vetor terá seus elementos deslocados uma posição à esquerda (na direção do início do vetor). A posição vaga no final do vetor após o deslocamento devem receber valor 0 (zero). Ao final, o programa deve imprimir o valor do elemento excluído e o conteúdo do vetor resultante. OBS.: Considere que a função void imprime_vetor(int vet[]) para mostrar conteúdo de um vetor JÁ EXISTE. Sua solução deverá apenas CHAMAR esta função onde necessário.

Exemplo de execução (com N=10):
Informe vetor: 5 6 8 2 1 3 9 4 7 10
Indique posicao a eliminar: 4
Vetor final: 5 6 8 2 3 9 4 7 10 0
Valor eliminado: 1

63) Uma companhia de eletricidade implantou uma política que reajusta a conta de energia elétrica daqueles que gastam além de um determinado valor limite. O reajuste varia de acordo com uma das seguintes situações: (i) reajuste de 10% para quem gastou até 40% além do valor limite; (ii) reajuste de 25% para quem gastou de acima de 40% e até 70% do valor limite; (iii) reajuste de 50% para quem gastou acima de 70% do valor limite. Faça um programa em C++ que obtenha um valor limite para cálculos de reajustes, e um vetor de N valores reais (N estabelecido via #define), cada valor representando o valor gasto por cada uma das N residências avaliadas. Atualize o valor das contas de cada residência no vetor caso elas ultrapassem o valor limite. Por fim, mostre os valores reajustados que cada residência deve pagar para saldar sua conta de energia e o total acumulado que será arrecadado a mais pela companhia elétrica. OBS.: Considere que a função void

imprime_vetor(float vet[]) para mostrar o conteúdo de um vetor JÁ EXISTE. Sua solução deverá apenas CHAMAR esta função onde necessário.

Exemplo de execução (com N=10):
Digite valor limite: 100
Digite contas de residencias: 20 40 60 80 100 120 140 160 180 200
Valores antes do reajuste: 20 40 60 80 100 120 140 160 180 200
Valores depois do reajuste: 20 40 60 80 100 132 154 200 270 300
Valor extra arrecadado: 256

Exemplo de execução (com N=5):
Digite valor limite: 300
Digite contas de residencias: 100 200 300 400 500
Valores antes do reajuste: 100 200 300 400 500
Valores depois do reajuste: 100 200 300 440 625
Valor extra arrecadado: 165

64) Faça um programa em C++ que leia um vetor de tamanho N (N estabelecido via #define) contendo inteiros x1,x2,...,xN representando o resultado da rodada de um campeonato de futebol com N times. Cada valor xk pode ser: (a) negativo (ou nulo), representando a diferença de gols pela qual o time k perdeu (ou empatou) o jogo dessa rodada; (b) positivo, representando o índice (de 1 a N) do time que foi adversário do time k nessa rodada. Por exemplo, para sequência -2 1 4 0, temos que: o time 1 perdeu o jogo por uma diferença de 2 gols, o time 2 jogou contra o time 1, o time 3 jogou contra o time 4 e o time 4 empatou a partida. Seu programa deve alterar o vetor lido de forma que toda posição passe a ter o saldo de gols do respectivo time nessa rodada. Para o exemplo, o 2o elemento deve ser alterado para 2 (pois o time 2 jogou contra o time 1 que tem um saldo de -2 gols) e o 3o elemento deve ser alterado para 0 (pois o time 3 jogou contra o time 4 que tem um saldo de 0 gols). Ao final, seu programa deve imprimir o vetor resultante, a maior diferença de gols que houve nas partidas da rodada e a quantidade de partidas que terminaram empatadas. Considere que todo time perdedor apresenta no vetor o seu saldo de gols e todo time ganhador apresenta o índice do seu adversário, variando de 1 a N. No caso de empate, apenas um deles tem o índice do adversário. OBS.: Considere que a função void imprime_vetor(int vet[]) para mostrar o conteúdo de um vetor JÁ EXISTE. Sua solução deverá apenas CHAMAR esta função onde necessário.

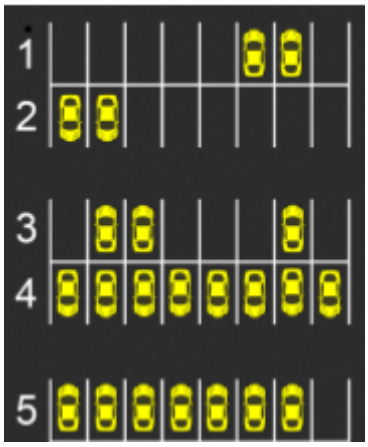
Exemplo de execução (com N=6):
Resultado da rodada: -2 6 1 5 -1 0
Saldo de gols:
-2 0 2 1 -1 0
Maior diferença de gols: 2
Qtd. de jogos empatados: 1

Outro exemplo de execução (com N=10):
Resultado da rodada: 0 0 -3 9 1 -3 6 2 -1 3
Saldo de gols: 0 0 -3 1 0 -3 3 0 -1 3
Maior diferença de gols: 3
Qtd. de jogos empatados: 2

65) Faça um programa em C++ que leia (do teclado) um vetor de tamanho N (N estabelecido via #define) de números inteiros e altere este vetor, de forma que todos os elementos divisíveis pelo primeiro valor do vetor recebam o novo valor -1. Ao final do processamento, o programa deverá mostrar o novo conteúdo do vetor no monitor de vídeo e informar a soma dos valores que foram substituídos por -1. OBS.: Considere que a função void imprime_vetor(int vet[]) para mostrar o vetor criado JÁ EXISTE. Sua solução deverá apenas CHAMAR esta função onde necessário.

Exemplo de execução (com N=10):
Informe vetor: 3 4 2 5 6 3 7 9 12 15
Vetor gerado: -1 4 2 5 -1 -1 7 -1 -1 -1
Soma: 48

66) Um estacionamento quer controlar seu movimento de carros e instalou sensores em cada uma de suas vagas. Os sensores informam, sempre que requisitado, se existe ou não um veículo estacionado na vaga, resultando em uma matriz com valores que representam vaga livre ou vaga ocupada.



Escreva um programa em linguagem C++ que receba uma matriz representando um estacionamento de M corredores, cada corredor com N vagas (M e N estabelecidos via #define), cada elemento da matriz assumindo valor 1 (vaga ocupada) ou 0 (vaga livre). O programa deve informar: (i) se existe ou não algum corredor totalmente ocupado (se existir, informar quais), e (ii) quantas vagas do estacionamento estão ocupadas e quantas estão livres. OBS.: Para a leitura de uma matriz, considere que já existe a função void ler_estacionamento (int matriz[][N]). Sua solução deve apenas chamar esta função.

Exemplo de execução (com M=5 e N=8):
Informe estacionamento:
0 0 0 0 0 1 1 0
1 1 0 0 0 0 0 0
0 1 1 0 0 0 1 0
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0
Corredor 4 está lotado.
Existem 22 vagas ocupadas e 18 livres.

67) Escreva um programa em C++ que deverá ler uma matriz com M linhas x N colunas (M e N estabelecidos via #define) de valores inteiros, onde cada elemento da matriz representa uma cota (em centímetros) do correspondente metro quadrado de um terreno de M×N metros que precisa ser aplainado. Ao final, o programa deve imprimir, como no exemplo abaixo, o volume de terra (em metros cúbicos) que será necessário adicionar (ou retirar) para fazer a terraplenagem de toda a área do terreno, visando colocar todo o terreno em uma cota uniforme de 128 centímetros.

OBS.: Para a leitura de uma matriz, considere que já existe a função void ler_matriz (int matriz[][N]). Sua solução deve apenas chamar esta função.

Exemplo de execução (com M=3 e N=6):
Informe a matriz original:
19 68 58 80 21 177
66 28 30 32 34 156
78 58 60 52 74 172
Qtde. de terra a acrescentar: 10.41 m3

Outro exemplo de execução (com M=3 e N=6):
Informe a matriz original:
219 168 158 180 125 217
166 128 130 232 234 136
278 158 160 152 214 152
Qtde. de terra a retirar: 9.03 m3

68) Teobaldo é proprietário de uma concessionária de carros e precisa da sua ajuda para verificar a produtividade de N funcionários das vendas semanais de um mês. Sua tarefa é escrever um programa em C++ que receba do usuário uma matriz de N linhas x 4 colunas (N estabelecido via #define) de valores reais, na qual cada linha representa um vendedor e as colunas são os totais (em reais) de vendas feitas por ele em cada semana do mês (considerando-se meses de 4 semanas). Ao final, seu programa deve mostrar na tela: a) o vendedor mais produtivo, b) o total do mês, c) a média dos vendedores. OBS.: Para a leitura de uma matriz, considere que já existe a função void ler_matriz (float matriz[][4]). Sua solução deve apenas chamar esta função.

Exemplo de execução (com N=3):
Informe as vendas:
30000 25450 12500 10000
11500 13550 21450 12999
40000 35000 15600 19500
Vendedor mais produtivo: 3
Total Mensal: R$ 247549.00
Média dos Vendedores: R$ 82516.33

69) Escreva um programa em C++ que faz a leitura de uma matriz quadrada de números inteiros de tamanho ímpar N ×N (N estabelecido via #define). O programa

deve alterar para valor 0 todas as posições cujo valor original seja maior que o VMLA (valor médio da linha anterior). Considere vmla = 5 para alterar os valores da primeira linha da matriz, pois ela não tem linha anterior. Ao final, o programa deve exibir na tela o novo conteúdo da matriz. OBS.: Para a leitura de uma matriz, considere que já existe a função void ler_matriz (int matriz[][N]). Sua solução deve apenas chamar esta função.

Exemplo de execução (com N=3):
Informe matriz:
2 8 3
6 1 4
4 7 8
Matriz resultante:
2 0 3
0 1 4
0 0 0

Outro exemplo de execução (com N=5):
Informe matriz:
1 2 8 3 9
0 6 1 4 7
9 4 7 8 2
2 7 1 3 8
1 8 2 5 9
Matriz resultante:
1 2 0 3 0
0 0 1 4 0
0 0 0 0 2
2 0 1 3 0
1 0 2 0 0

70) Escreva um programa em C++ que deverá ler uma matriz com M linhas x N colunas (M e N estabelecidos via #define) de valores inteiros, onde cada elemento da matriz representa uma cota (em centímetros) do correspondente metro quadrado de um terreno de M×N metros que precisa ser aplainado. Ao final, o programa deve imprimir, como no exemplo abaixo, o volume de terra (em metros cúbicos) que será necessário adicionar (ou retirar) para fazer a terraplenagem de toda a área do terreno, visando colocar todo o terreno em uma cota uniforme de 128 centímetros. OBS.: Para a leitura de uma matriz, considere que já existe a função void ler_matriz (int matriz[][N]). Sua solução deve apenas chamar esta função.

Exemplo de execução (com M=3 e N=6):
Informe a matriz original:
19 68 58 80 21 177
66 28 30 32 34 156
78 58 60 52 74 172
Qtde. de terra a acrescentar: 10.41 m3

Outro exemplo de execução (com M=3 e N=6):
Informe a matriz original:
219 168 158 180 125 217
166 128 130 232 234 136
278 158 160 152 214 152
Qtde. de terra a retirar: 9.03 m3

71) Escreva um programa em C++ que deverá ler uma matriz com M linhas x N colunas (M e N estabelecido via #define) de valores reais. O programa deverá calcular o número de linhas e o número de colunas nulas da matriz, estes valores devem ser mostrados na tela. Além disso, os valores da matriz devem ser divididos pelo número de linhas nulas. Ao final, a matriz resultante deve ser impressa na tela. OBS.: Para a leitura de uma matriz, considere que já existe a função void le_matriz (float matriz[][N]), que preencherá toda a matriz. Sua solução deve apenas chamar esta função.

Exemplo de execução (com M=4 e N=4):
Informe matriz:
1 0 2 3
4 0 5 6
0 0 0 0
0 0 0 0
Matriz tem 2 linhas e 1 coluna nulas
Matriz resultante:
0.5 0 1 1.5
2 0 2.5 3
0 0 0 0
0 0 0 0

72) O município de Blumenau, em Santa Catarina, para proteger os cidadãos de calamidades devidas a enchentes, monitora o nível (em metros) do rio Itajaí-Açu de tempos em tempos, e criou um sistema de alerta à população.

Todos os locais do município suscetíveis a enchentes tiveram suas cotas (em metros), em relação ao fundo do rio, mapeados e registrados. Foram também identificados e catalogados abrigos para os quais a população deve deslocar-se em caso de enchente. Para cada local do município sujeito a enchentes foram registrados o código do local, sua cota (em metros) em relação ao fundo do rio, o código do abrigo para onde seus moradores devem deslocar-se em caso de enchente e o número de avisos emitidos para aquele local (inicialmente com valor zero e incrementado de 1 a cada processamento do programa com situação de enchente detectada). Escrever um programa em C++ que leia do usuário o nível atual do rio e uma matriz de monitoramento de 4 linhas e N colunas (N estabelecido via #define), cada coluna correspondente aos dados dos locais do município sujeitos a enchentes e cada linha correspondente a código do local, cota, código de abrigo e número de avisos, conforme descrito anteriormente. O programa deve mostrar, para os locais com cotas em situação de enchente, os locais, cotas e abrigos e, ainda, incrementar o número de avisos emitidos para aquele local na matriz. OBS.: Para a leitura de uma matriz, considere que já existe a função void le_matriz (float matriz[] [N]), que preencherá toda a matriz. Sua solução deve apenas chamar esta função.

Exemplo de execução (para N = 9):
Nível do rio (m): 10.05
Matriz de monitoramento:

30	31	32	33	34	35	36	37	40
9	9.3	9.5	9.7	9.5	11	12.3	10.9	11
90	90	90	91	91	92	55	55	55
3	2	2	1	2	2	1	0	0

Enchentes:

Locais	Cotas	Abrigos
30	9	90
31	9.3	90
32	9.5	90
33	9.7	91
34	9.5	91

Matriz de monitoramento final:

30	31	32	33	34	35	36	37	40
9	9.3	9.5	9.7	9.5	11	12.3	10.9	11
90	90	90	9	91	92	55	55	55
4	3	3	2	3	2	1	0	0

73) Um comerciante deseja catalogar os M produtos que comercializa em uma matriz de M linhas. Para cada produto, ele deseja registrar como colunas (N) da matriz: o código, a quantidade em estoque, o custo unitário de aquisição e o preço unitário de venda praticado. Escrever um programa em C++ que permita ao comerciante desenvolver o cadastro desejado e, a partir deste, determinar o custo total de seu estoque, o item com maior rentabilidade absoluta (obtida pela diferença entre preço e custo unitários) e sua perspectiva de ganho se vender todo o estoque deste item. Os valores de M e N deverão ser estabelecidos via diretiva #define. OBS.: Para a leitura de uma matriz, considere que já existe a função void ler_matriz (float matriz[][N]). Sua solução deve apenas CHAMAR esta função.

Exemplo de execução (com M=8, N=4):
CATÁLOGO DE PRODUTOS

Cód.	Estoque	Custo	Preço
.00010	35000	.025	.05
.01023	135	7.134	9.5
.23189	3345	2.74	5
.00235	15	115.285	209
.12131	52356	.3598	.85
.00123	854	12.557	13.3
.00020	2600	.472	1
.00025	1000	.125	.50

Custo total do estoque: 43646.2318
Item com maior rentabilidade: .00235
Lucro bruto previsto: 1405.725

74) Faça um programa em C++ que leia uma matriz M × xN de números inteiros (M e N definidos via diretiva #define) e calcula qual é a linha cuja soma é maior que a soma das outras linhas, informando este valor ao usuário. Caso haja mais que uma linha com soma maior que as outras, mostrar apenas a primeira que satisfaz a condição. OBS.: Para a leitura de uma matriz, considere que já existe a função void ler_matriz (int matriz[][N]). Sua solução deve apenas CHAMAR esta função.

Exemplo de execução (com M=4 e N=3):
Informe a matriz:

8	2	7
4	5	5
3	10	2

5 10 1
linha 0, soma 17

75) O gabarito de cada participante de uma prova de concurso foi digitalizado e uma matriz de números inteiros correspondente foi gerada. Essa matriz tem uma linha para cada questão e uma coluna para cada alternativa. Assim, cada posição contém o valor 0 se a alternativa não foi marcada pelo participante e 1 caso contrário. Faça um programa em C++ que leia um vetor de tamanho N, onde cada posição i contém um char representando a alternativa correta da i-ésima questão (‘A’ se a primeira alternativa é a correta, ‘B’ se a segunda, e assim por diante). Em seguida, seu programa deve ler uma matriz de dimensões N × M representando as repostas de um participante e, por fim, deve imprimir a quantidade de questões que o participante acertou. Considere que questões com 2 ou mais alternativas marcadas estão incorretas. Os valores N e M devem ser estabelecidos via #define.
Dica: O código ASCII do caractere ‘A’ é 65, do ‘B’ é 66, e assim por diante. OBS.: Para a leitura da matriz e do vetor, considere que já existem as funções void le_matriz(int mat[][M]) e void le_vetor(int vet[]). Seu programa deve apenas CHAMAR estas funções.

Exemplo de execução (com N=5, M=4):
Entre com o vetor:
A A D B C
Entre com o gabarito:
0 0 1 0
1 0 0 0
1 0 0 1
0 1 0 0
0 0 0 1
O participante acertou 2 questões.

76) Faça um programa em C++ que crie uma matriz simétrica. O programa deve calcular a soma dos valores que estão acima da diagonal principal e a soma dos valores que estão abaixo da diagonal e compará-los. A parte que possuir a maior soma será espelhada na outra parte da matriz. Considere para essa operação que a matriz utilizada será quadrada de dimensões N linhas e N colunas, (N estabelecido via diretiva #define). Neste programa deve ser criada e usada uma função chamada

matriz_simetrica que recebe como argumento a matriz a ser transformada em simétrica. Essa função deve calcular a soma dos valores acima da diagonal principal, a soma dos valores abaixo da diagonal principal e espelhar a matriz original com os valores da parte que tiver a maior soma na outra parte da matriz, a partir da diagonal principal. OBS.: Para obter os dados da matriz e para imprimir os dados da matriz, considere que já existem as funções void ler_matriz (int matriz[][N]) e void imprimir_matriz(int mat[][N]) respectivamente, já codificadas e disponíveis para uso. Sua solução deve apenas chamar estas funções onde necessário.

Exemplo de execução (para N = 4):
Entre com a matriz:
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
Soma acima da diagonal principal: 36
Soma abaixo da diagonal principal: 66
Matriz simétrica:
1 5 9 13
5 6 10 14
9 10 11 15
13 14 15 16

77) Fazer programa em C++ que lê uma matriz quadrada N × N, (N estabelecido via diretiva #define), calcula a soma dos valores na diagonal principal e a soma dos valores na diagonal secundária, imprimindo na tela o resultado. O cálculo deve ser feito por uma função que recebe a matriz e retorna os dois resultados. OBS.: Para a leitura de uma matriz, considere que já existe a função void ler_matriz (int matriz[] [N]). Sua solução deve apenas chamar esta função onde necessário.

Exemplo de execução (com N=3):
Informe Matriz:
3 4 5
2 3 6
7 1 2
Soma diagonal principal = 8
Soma diagonal secundaria = 15

78) Em um engradado de bebidas, pode-se ter: Água (1), refrigerante (2), cerveja (3) ou nada (0). Um refrigerante possui, em média, 10,8 gramas de açúcar para cada 100 ml de bebida, uma cerveja tem em média 5% de álcool na sua composição e a água não possui nada de álcool ou açúcar. Dado um engradado de bebidas, representado por uma matriz M × N, sendo os valores de M e N estabelecidos pela diretriz #define, escreva um programa em C++ que leia a matriz e utilize uma função que receba a matriz e calcule: o número total de garrafas, o total de litros de bebida; o total de gramas de açúcar; o total em litros de álcool; e a concentração por litro de açúcar em todo o engradado. Todas as garrafas do engradado possuem a mesma capacidade e ela é fornecida pelo usuário no início do programa. OBS.: Para a leitura da matriz que representa o engradado, considere que já existe a função void ler_matriz (int matriz[][N]) já codificada e disponível para uso. Sua solução deve apenas chamar estas funções onde necessário.

Exemplo de execução (para M=2 e N=6):
Quantos litros têm as garrafas? 0.6
Entre com o engradado:
1 2 3 3 2 0
0 0 2 2 1 1
Total de garrafas: 9
Total de líquido: 5.4 l
Total de açúcar: 259.2 g
Total de álcool: 0.06 l
Concentração total de açúcar: 48 g/l.

Outro exemplo de execução (para M=2 e N=6):
Quantos litros têm as garrafas? 1
Entre com o engradado:
3 3 3 3 2 3
0 0 2 2 1 1
Total de garrafas: 10
Total de líquido: 10 l
Total de açúcar: 324 g
Total de álcool: 0.25 l
Concentração total de açúcar: 32.4 g/l.

79) Faça um programa em C++ que leia uma matriz M × N de números inteiros (M e N estabelecidos via #define), calcule e imprima quantos elementos 2-maior existem nessa matriz. Um elemento é 2-maior caso seja maior que os seus vizinhos de linha

(da direita e da esquerda). Caso não exista vizinho, considere que este possui o valor 0 (zero). Para resolver o problema proposto pede-se o desenvolvimento e uso de uma função de nome calcula_2maior() que recebe como entrada uma matriz, calcula e retorna o número de elementos 2-maior. OBS.: Para a leitura de uma matriz, considere que já existe a função void ler_matriz (int matriz[][N]). Sua solução deve apenas CHAMAR esta função.

Exemplo de execução (com M=3, N=4):
Informe a matriz
7 15 4 11
5 3 20 0
1 6 9 8
Qtde. elementos 2-maior: 5

80) Escrever um programa em C++ que leia uma matriz de números reais de ordem M × N (M e N estabelecidos via #define, sendo N > 2), verifique e mostre as linhas nas quais os dois primeiros valores da linha e qualquer dos valores restantes da mesma linha podem formar um triângulo. Indicar os casos de linhas que não permitam a formação de triângulo algum. Para a formação de um triângulo, a soma de dois lados quaisquer deve ser maior que o terceiro. OBS.: Para a leitura de uma matriz, considere que já existe a função void ler_matriz (float matriz[][N]). Sua solução deve apenas CHAMAR esta função.

Exemplo de execução (com M=5, N=4):
Indique a matriz:
3 4 5 8
0 0 0 0
1 2 3 4
4 3 2 3
1 1 1 3
Resultado:
Linha 0 colunas 0 e 1 com 2
Linha 1 não permite formar triângulos
Linha 2 não permite formar triângulos
Linha 3 colunas 0 e 1 com 2 e 3
Linha 4 colunas 0 e 1 com 2

81) Faça um programa em C++ que leia uma matriz M × N de números inteiros (M e N estabelecidos via #define) e: imprime Borda se a soma dos elementos da borda

da matriz é maior que soma dos elementos do miolo da matriz; ou imprime Miolo caso contrário. A borda da matriz são todos os elementos das primeiras e últimas colunas e linhas; o miolo são todos os outros elementos da matriz. Você deve escrever a função MioloMoleCascaDura(), que recebe uma matriz M×N e retorna 1 (um) se a soma dos elementos da borda da matriz é maior ou igual do que a soma dos elementos do miolo e 0 (zero) caso contrário. OBS.: Para a leitura de uma matriz, considere que já existe a função void ler_matriz (int matriz[][N]). Sua solução deve apenas CHAMAR esta função.

Exemplo de execução (com M=3, N=4):
Informe matriz:
1 4 2 5
23 5 4 9
9 3 7 5
Borda

Outro exemplo de execução (com M=3, N=4):
Informe matriz:
1 4 1 5
1 25 14 2
2 3 7 5
Miolo

82) Faça um programa em C++ que leia uma matriz quadrada M ×M de números inteiros (M estabelecido via #define), calcule e imprima a soma dos menores elementos de cada linha e de cada coluna da matriz. Crie e utilize no programa a função smenorColLin(), que recebe uma matriz M × M e um valor k e retorna a soma do menor elemento da linha k com o menor elemento da coluna k da matriz. OBS.: Para a leitura de uma matriz, considere que já existe a função void ler_matriz (int matriz[][M]). Sua solução deve apenas CHAMAR esta função.

Exemplo de execução (com M=4):
Informe matriz:
1 4 2 5
23 5 4 9
9 3 7 5
6 5 4 2
Soma dos menores: 18

83) Fazer programa em C++ que lê uma matriz M×N (M e N estabelecidos via diretiva #define) a partir do teclado, e informa se a matriz lida atende o Modelo de Vandermonde Totalmente, Parcialmente, ou não Atende. A avaliação deve ser feita por uma função chamada Avalia_Vandermonde (que deverá ser codificada junto com o resto do código) que recebe a matriz e retorna o valor 1 (indicando que atende Totalmente), o valor 0 (indicando que atende Parcialmente) ou o valor -1 (indicando que não Atende Vandermode). Uma matriz atende totalmente o Modelo de Vandermonde quando M for igual a N e cada item da matriz está dentro do Modelo abaixo. Uma matriz atende parcialmente o modelo de Vandermonde quando está dentro do modelo abaixo, mas $M \neq N$. E a matriz não atende o modelo de Vandermonde quando não atende o modelo abaixo com quaisquer dos valores da matriz. No modelo abaixo, $X_0, X_1, \cdots X_{M-1}$ são os valores básicos da matriz e podem possuir valores repetidos (por exemplo, X_0 pode ser igual a X_6).

$$\begin{vmatrix} 1 & {X_0}^1 & {X_0}^2 & {X_0}^3 & {X_0}^4 & \cdots & {X_0}^{N-1} \\ 1 & {X_1}^1 & {X_1}^2 & {X_1}^3 & {X_1}^4 & \cdots & {X_1}^{N-1} \\ 1 & {X_2}^1 & {X_2}^2 & {X_2}^3 & {X_2}^4 & \cdots & {X_2}^{N-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & {X_{M-1}}^1 & {X_{M-1}}^2 & {X_{M-1}}^3 & {X_{M-1}}^4 & \cdots & {X_{M-1}}^{N-1} \end{vmatrix}$$

OBS.: Para a leitura de uma matriz, considere que já existe a função void ler_matriz (int matriz[][N]). Sua solução deve apenas CHAMAR esta função.

Exemplo de execução (com M=3, N=3):
Digite matriz 3x3
3 4 5
2 3 6
7 1 2
Matriz não atende Vandermonde

Outro exemplo de execução (com M=3, N=3):
Digite matriz 3x3
1 4 16
1 3 9
1 2 4
Matriz atende Vandermonde totalmente

Outro exemplo de execução (com M=2, N=3):
Digite matriz 2x3
1 4 16
1 7 49

Matriz atende Vandermonde parcialmente

84) Faça um programa em C++ que leia uma matriz quadrada N × N de números inteiros (N estabelecido via #define). Lida a matriz, o programa deve mostrar o valor da soma dos elementos acima da diagonal e o conteúdo da matriz. Para produzir estes resultados, deve ser criada e usada no programa a função de nome calculo_matriz que recebe como argumento uma matriz N × N e retorna a soma dos valores que estão acima da diagonal principal, além de alterar o valor de todos os elementos da mesma matriz, multiplicando cada valor original da matriz pelo valor da soma calculada. OBS.: Para a leitura de uma matriz, considere que já existe a função void ler_matriz (int matriz[][N]). Para imprimir os valores da matriz na tela, considere que já existe a função void imprimir_matriz (int matriz[][N]). Sua solução deve apenas CHAMAR estas funções onde apropriado.

Exemplo de execução (com N=4):
Informe matriz:
1 0 0 0
5 6 1 1
9 2 3 0
1 0 4 8
Soma acima da diagonal princ.: 2
Matriz final:
2 0 0 0
10 12 2 2
18 4 6 0
2 0 8 16

85) Faça um programa em C++ que leia a partir do teclado uma matriz M × N de números inteiros (M e N estabelecido via #define)representando notas de escolas de samba. A matriz possui uma linha para cada escola e uma coluna para cada jurado. O programa deve descartar a menor nota atribuída pelos jurados às escolas de samba e calcular a quantidade de escolas nota 10. Uma escola é nota 10 se possui todas as notas iguais a 10 (dez) após o descarte. Para resolver o problema proposto pede-se o desenvolvimento e uso de uma função de nome apuracao() que recebe como argumento uma matriz M × N de números inteiros, substitui a menor nota de cada escola pelo valor 0 (zero), calcula e retorna a quantidade de escolas

nota 10. OBS.: Para a leitura de uma matriz, considere que já existe a função void ler_matriz (int matriz[][N]). Para imprimir os valores da matriz na tela, considere que já existe a função void imprimir_matriz (int matriz[][N]). Sua solução deve apenas CHAMAR estas funções onde apropriado.

Exemplo de execução (com M=3 e N=4):
Informe matriz:
7 10 9 10
10 10 10 8
10 10 10 10
Menores notas descartadas:
0 10 9 10
10 10 10 0
0 10 10 10
Qtde. escolas nota 10: 2