Curso de Programação

Professor Ricardo Demattê

IFSP – Unidade Salto

Aula 1 – Conceitos Básicos

Professor Ricardo Demattê

IFSP – Unidade Salto

INSTITUTO FEDERAL
São Paulo
Câmpus Salto

Introdução a linguagem C e C++

Características da Linguagem C e C++

- C ++ é uma das linguagens de programação mais populares do mundo.
- C ++ pode ser encontrado nos sistemas operacionais atuais, nas interfaces gráficas do usuário e nos sistemas embarcados.
- C ++ é uma linguagem de programação orientada a objetos que fornece uma estrutura clara aos programas e permite que o código seja reutilizado, reduzindo os custos de desenvolvimento.
- C ++ é portátil e pode ser usado para desenvolver aplicativos que podem ser adaptados a várias plataformas.
- C ++ é divertido e fácil de aprender!
- Como C ++ é próximo a C # e Java, torna mais fácil para os programadores mudarem para C ++ ou vice-versa

Introdução a linguagem C e C++

Características da Linguagem C e C++

- Em proposta geral, C++ deve ser tão eficiente e portável quanto o C
- C++ é desenvolvido para ser o quanto mais compatível com C possível
- C++ é desenvolvido para suportar múltiplos paradigmas de programação,
- C ++ é uma linguagem que pode ser usada para criar aplicativos de alto desempenho.
- C ++ foi desenvolvido por Bjarne Stroustrup, como uma extensão da linguagem C.
- C ++ oferece aos programadores um alto nível de controle sobre os recursos do sistema e a memória.
- A linguagem foi atualizada três vezes em 2011, 2014 e 2017 para C ++ 11, C ++ 14 e C ++ 17.

O desenvolvimento da linguagem C

Ken Thompson e Dennis Ritchie,
os criadores das linguagens B e C.

- O desenvolvimento inicial de C ocorreu no AT&T Bell Labs entre 1969 e 1973 como uma linguagem de implementação de sistema para o nascente sistema operacional Unix.
- Não se sabe se o nome "C" foi dado à linguagem porque muitas de suas características derivaram da linguagem B e C é a letra conseguinte no alfabeto, ou porque "C" é a segunda letra do nome da linguagem BCPL, da qual derivou-se a linguagem B.

Evolução: A linguagem C++

Bjarne Stroustrup é um cientista da computação dinamarquês e professor catedrático da Universidade do Texas A&M. É conhecido como o pai da linguagem de programação C++

- No final da década de 1970, a linguagem C começou a substituir a linguagem BASIC como a linguagem de programação de microcomputadores mais usada. Durante a década de 1980, foi adaptada para uso no PC IBM, e a sua popularidade começou a aumentar significativamente. Ao mesmo tempo, Bjarne Stroustrup, juntamente com outros nos laboratórios Bell, começou a trabalhar num projeto onde se adicionavam construções de linguagens de programação orientada por objetos à linguagem C.
- A linguagem que eles produziram, chamada C++, é nos dias de hoje a linguagem de programação de aplicações mais comum no sistema operacional Windows da companhia Microsoft; C permanece mais popular no mundo Unix.

Aula 2 – Compiladores

Professor Ricardo Demattê

IFSP – Unidade Salto

Conceitos básicos

- Para começar a usar C ++, você precisa de duas coisas:

- Um **editor de texto**, como o Notepad, para escrever código C ++
- Um **compilador**, como o GCC, para traduzir o código C ++ em uma linguagem que o computador irá entender.

- Toda linguagem de programação possui um tradutor de código. Este tradutor pode ser um compilador ou um interpretador, dependendo da linguagem.
- Interpretadores são programas que leem o código-fonte e executam ele diretamente, sem a criação de um arquivo executável.
- Chamamos de compilador o programa que traduz um arquivo escrito em código de linguagem de programação (arquivo-fonte) para a linguagem do microprocessador, criando um arquivo capaz de executar as instruções pedidas (arquivo executável).

Notepad ++

- **Notepad++**
  - Notepad++ é um editor de texto e de código fonte de código aberto sob a licença GPL. Suporta várias linguagens de programação rodando sob o sistema Microsoft Windows. O Notepad++ é distribuído como um Software livre. Rodando no ambiente MS Windows, seu uso é regido pela GNU General Public License.
- **Instalação**
  - A versão mais recente do NotepadC++ pode ser baixada através da página:
  - https://notepad-plus-plus.org/downloads/

DevC++

DevC++

- O Dev-C++ é um compilador freeware das linguagens C e C++. É uma opção muito interessante, pois é de fácil utilização e aprendizado para usuários novos e possui muitos recursos avançados para usuários experientes. Além de, claro, seu download ser gratuito.

Instalação

- A versão mais recente do DevC++ pode ser baixada através da página:

http://www.bloodshed.net/dev/devcpp.html, no link “Download”.

- Utilizou-se, na elaboração desta apostila, a versão do DevC++ beta 9.2, disponível diretamente através do link:

http://prdownloads.sourceforge.net/dev-cpp/devcpp-4.9.9.2_setup.exe

- Outros pacotes e bibliotecas do C++ podem ser encontradas no site

http://devpaks.org/

Code::Blocks

Code::Blocks

- Code::Blocks (ou C::B) é um ambiente de desenvolvimento integrado de código aberto e multiplataforma. Ele foi desenvolvido em C++, usando wxWidgets. Sua arquitetura é orientada a plugin, de forma que suas funcionalidades são definidas pelos plugins fornecidos a ele. Code::Blocks é voltado para o desenvolvimento em C/C++ e Fortran, podendo também ser usado para a criação de ARM, AVR, D (linguagem de programação), DirectX, FLTK, GLFW, GLUT, GTK+, Irrlicht, Lightfeather, MATLAB, OGRE, OpenGL, Qt, SDL, SFML, STL, SmartWin e programas ou aplicativos com wx, embora, em certos casos, a instalação de SDKs ou frameworks seja necessária.

Instalação

- A versão mais recente do Code:Blocks pode ser baixada através da página:

http://www.codeblocks.org/downloads/26

Code::Blocks

VS Code

VS Code

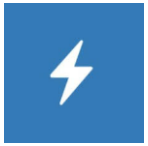
- O Visual Studio Code é um editor de código-fonte desenvolvido pela Microsoft para Windows, Linux e macOS. Ele inclui suporte para depuração, controle de versionamento Git incorporado, realce de sintaxe, complementação inteligente de código, snippets e refatoração de código.
- O Visual Studio Code suporta um número de linguagens de programação e um conjunto de recursos que podem ou não estarem disponíveis para a dada linguagem, como mostrado na tabela a seguir.

Instalação

- A versão mais recente do VS Code pode ser baixada através da página:

https://code.visualstudio.com/download, no link “Download”.

Compiladores on-line

- **Online GBD** 

  - GDB online é um compilador online e ferramenta de depuração para C, C ++, Python, PHP, Ruby, C #, VB, Perl, Swift, Prolog, Javascript, Pascal, HTML, CSS, JS. Codifique, compile, execute e depure online de qualquer lugar do mundo.
  - Acesse: https://www.onlinegdb.com/

- **Replit** 

  - O Replit permite que os usuários escrevam códigos e criem aplicativos e sites usando um navegador. O site também possui vários recursos colaborativos, incluindo capacidade de edição multiusuário em tempo real com feed de chat ao vivo. Ele oferece suporte a mais de 50 linguagens de programação e marcação, incluindo Java, Python e HTML, permitindo que os usuários criem aplicativos e sites. O site é integrado ao GitHub, uma plataforma de hospedagem de código, que fornece uma maneira de importar e executar projetos no GitHub.
  - Acesse: https://replit.com/

Aula 3 – Sintaxe de um Programa

Professor Ricardo Demattê

IFSP – Unidade Salto

Estrutura básica de um programa

- Estrutura de um programa na **linguagem C**:

```c
// exemplo

#include <stdlib.h>                    // biblioteca  padrão do C (salvar como *.c)
#include <stdio.h>                     // biblioteca  padrão do C (salvar como *.c)

main()                                 // função principal
{                                      // início do programa
     //comandos do programa            // comentário
     system("PAUSE > null");           // pausa o programa
     return 0;                         // resposta de erros da função principal
}                                      // início do programa
```

- Estrutura de um programa na **linguagem C++:**

```c
// exemplo

#include <iostream>                    // biblioteca  padrão do C++ (salvar como *.cpp)
using namespace  std;                  // padrão de nomenclatura do C++

main()                                 // função principal
{                                      // início do programa
     // comandos do programa           // comentário
     system("PAUSE > null");           // pausa (não funciona em compilador on-line
     return 0;                         // resposta de erros da função principal
}                                      // início do programa
```

Bibliotecas do C

#include <stdlib.h>

#include <stdio.h>

- As duas primeiras linhas são o cabeçalho do programa. Todo programa deve ter um cabeçalho desse tipo para definir quais as bibliotecas ele utilizará. “Bibliotecas” são arquivos que normalmente são instalados juntos com o compilador e que possuem os comandos e funções pertencentes à linguagem.
- O cabeçalho **#include<>** serve para indicar ao compilador todas as bibliotecas que este programa utilizará.
- A primeira linha do cabeçalho, temos **#include <stdlib.h>.** Esta é uma biblioteca muito utilizada. Esta biblioteca contém várias funções e constantes básicas. Quase todos os programas C usam essa biblioteca. Esta biblioteca é de propósito geral padrão da linguagem de programação C.
- Na maioria dos programas que escreveremos durante esta apostila, utilizaremos o **#include <stdio.h>**, que serve para incluir a biblioteca que contém os comandos básicos da linguagem C, como printf e scanf. Esta biblioteca também contém as principais funções, comandos e classes de entrada e saída de C.

Bibliotecas do C+

```
#include <iostream>

using namespace  std;
```

- As duas primeiras linhas são o cabeçalho do programa. Todo programa deve ter um cabeçalho desse tipo para definir quais as bibliotecas ele utilizará. “Bibliotecas” são arquivos que normalmente são instalados juntos com o compilador e que possuem os comandos e funções pertencentes à linguagem.
- O cabeçalho **#include<>** serve para indicar ao compilador todas as bibliotecas que este programa utilizará. Na maioria dos programas que escreveremos durante esta apostila, só utilizaremos o **#include <iostream>**, que serve para incluir a biblioteca iostream em nossos programas. Esta biblioteca contém as principais funções, comandos e classes de entrada e saída de C++.
- A segunda linha do cabeçalho, **using namespace std**; , é um aviso ao compilador que estaremos utilizando os comandos e funções padrão de C++. Assim, sempre que utilizamos um comando próprio de C++, o compilador reconhecerá automaticamente este comando como sendo pertencente à biblioteca padrão de C++.

Função principal

```
main ( )
  {


  }
```

- Assim como em C, tudo o que acontece durante a execução do programa está contido dentro de uma função principal, chamada main. Declaramos a função main com: main ( )
- Todos os comandos executados pelo programa estão contidos entre as chaves “{ }” da função main. Mais adiante estudaremos as funções à fundo e veremos que um programa pode ter mais de uma função, mas é indispensável que todos os programas possuam a função main.
- Todas as linhas de comando dentro da função main devem ser encerradas com um **; (ponto e virgula)**

Pause

system("PAUSE > null");

- Cada programa terá seus próprios comandos, logicamente. Entretanto, o encerramento de um programa geralmente é feito da mesma maneira para todos eles. As duas últimas linhas antes do fecha-chaves são dois comandos normalmente utilizados ao fim de um programa.
- A linha “system(“PAUSE > null”)” é uma chamada de função própria de C++. A função system( ) recebe argumentos como o PAUSE que na verdade são comandos para o sistema operacional.
- Neste caso, ela recebe o comando “PAUSE > null” para pausar a execução do programa até que o usuário aperte uma tecla qualquer. Utilizamos este recurso para que a tela do programa não seja terminada automaticamente pelo sistema, impedindo que vejamos os resultados do programa.
- Também pode ser usado somente o “PAUSE”.
- **OBS:** O comando SYSTEM **não pode** ser utilizado em compiladores on-line.

Return

return 0;

- Finalmente, o comando “return 0” é a “resposta” da função main para o sistema. Quase toda função retorna um valor para o sistema ou programa que a chamou, no caso da função main, ela retorna um valor para o sistema operacional que executou o programa.
- Esse valor é interpretado pelo sistema como uma mensagem indicando se o programa foi executado corretamente ou não. Um valor de retorno 0 indica que o programa foi executado sem problemas; qualquer outro valor de retorno indica problemas.
- Quando o programa é executado até o fim, ele retorna 0 ao sistema operacional, indicando que ele foi executado e terminado corretamente. Quando o programa encontra algum erro ou é terminado antes da hora, ele retorna um valor qualquer ao sistema, indicando erro durante a execução.

Primeiro programa em C

// exemplo do **Hello World**

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

main()
{
    printf ("Ola' Mundo !!!");
    system("PAUSE > null");  //função opcional
    return 0;
}
```

ou para os compiladores on-line:

```c
#include <stdio.h>

int main()
{

    printf("Olá mundo !!!\n");
    return 0;
}
```

Primeiro programa em C++

// exemplo do **Hello World**

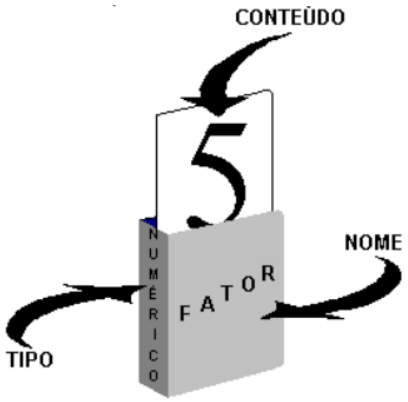
```
1   #include <iostream>
2
3   using namespace std;
4
5   main()
6   {
7       cout<<"Hello World";
8       system ("PAUSE >> null");
9       return 0;
10  }
```

ou

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main()
6   {
7       cout<<"Hello world !!!\n";
8       return 0;
9   }
```

Variáveis

- Uma **variável é uma informação que fica armazenada na memória**. Pode ser um número, um texto, dentre outras coisas.
- A linguagem C nos permite declarar variáveis, que possuem um tipo determinado. Declarar uma variável é como dar um nome a um pedaço da memória que guardará um valor de determinado tipo. Daí em diante podemos acessar o valor guardado e manipulá-lo através do nome da variável. Para declarar uma variável de tipo int e nome x, fazemos assim: **int x;**
- Na figura abaixo, a caixa (variável) rotulada com **FATOR** contém o valor **5**. Como seu tipo é **numérico**, em um determinado instante essa caixa poderá conter qualquer valor numérico (inteiro ou fracionário; positivo, negativo ou zero). Entretanto, em um determinado instante, ela conterá um, e somente um, valor.



Declaração de variáveis

- Para declarar uma variável somente é obrigatório declarar seu tipo e nome:

<tipo> <nome>;

Por exemplo: **int exemplo;**

- Além disso, caso seja necessário, podemos declarar um valor a esta variável no momento de sua declaração, e também adicionar um prefixo a ela, da seguinte forma:

<prefixo> <tipo> <nome> = <valor>;

Por exemplo: **int exemplo = 12;**

Nomes para variáveis

- **Existem algumas regras para a escolha dos nomes (ou identificadores) de variáveis em C++:**

  - Nomes de variáveis só podem conter letras do alfabeto, números e o caractere underscore “_”.
  - Não podem começar com um número.
  - Nomes que comecem com um ou dois caracteres underscore (“_” e “__”) são reservados para a implementação interna do programa e seu uso é extremamente desaconselhado. O compilador não acusa erro quando criamos variáveis desse jeito, mas o programa criado se comportará de forma inesperada.
  - Não é possível utilizar palavras reservadas da linguagem C++ (ver próximo slide). Também não é possível criar uma variável que tenha o mesmo nome de um função, mesmo que essa função tenha sido criada pelo programador ou seja uma função de biblioteca.
  - C++ diferencia letras maiúsculas e minúsculas em nomes de variáveis. Ou seja, count, Count e COUNT são três nomes de variáveis distintos.
  - C++ não estabelece limites para o número de caracteres em um nome de variável, e todos os caracteres são significantes.

Palavras reservadas

- Na linguagem C++ existem palavras que são de uso reservado, ou seja, que possuem funções específicas na linguagem de programação e não podem ser utilizadas para outro fim, como por exemplo, ser usada como nome de variável. Por exemplo, a palavra reservada “for” serve para chamar um laço de repetição, e não pode ser utilizada como nome de uma variável.
- A lista abaixo relaciona as palavras reservadas da linguagem C++:

abstract	assert	boolean	break	byte	case
catch	char	class	continue	default	do
double	else	enum	extends	false	final
finally	float	for	if	implements	import
instanceof	int	interface	long	native	new
null	package	private	protected	public	return
short	static	strictfp	super	switch	synchronized
this	throw	throws	transient	true	try
void	volatile	while			

Códigos de barra invertida

- **Constantes de barra invertida**
  - A linguagem C++ fornece constantes caractere mais barra invertida especiais, Estes códigos são mostrados na tabela a seguir:

Código	Significado
\b	Retrocesso (backspace)
\f	Alimentação de Formulário (form feed)
\t	Tabulação Horizontal (tab)
\n	Nova Linha
\”	Aspas
\’	Apostrofo
\0	Nulo
\\	Barra Invertida
\a	Sinal Sonoro (Beep)
\N	Constante Octal (N é o valor da constante)
\xN	Constante Hexadecimal (N é o valor da constante)

Operadores

- Um operador é um símbolo que diz ao compilador para realizar manipulações matemáticas e lógicas específicas.

Operador de atribuição

- O operador **“=”** atribui um valor ou resultado de uma expressão contida a sua direita para a variável especificada a sua esquerda.

Exemplos:

```
a = 10;
b = c * valor + getval(x);
a = b = c = 1;
```

- O último exemplo é interessante por mostrar que é possível associar vários operadores de atribuição em sequência, fazendo com que todas as variáveis envolvidas tenham o mesmo valor especificado.

Operadores

- **Operadores aritméticos**
  - São aqueles que operam sobre números e expressões, resultando valores numéricos. São eles:

Operador	Ação
+	Soma
-	Subtração ou troca de sinal
*	Multiplicação
/	Divisão
%	Resto da divisão inteira
++	Incremento
--	Decremento

Operadores

- **Operadores relacionais**
  - Operam sobre expressões, resultando valores lógicos de TRUE (verdadeiro) ou FALSE (falso). São eles

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

  - Atenção! Não existem os operadores relacionais: “=<“, “=>“ e “<>“. Não confunda a atribuição (“=“) com a comparação (“==“)!

Operadores

Operadores lógicos

- Operam sobre expressões, resultando valores lógicos de TRUE (verdadeiro) ou FALSE (falso). Possuem a característica de “short circuit”, ou seja, sua execução é curta e só é executada até o ponto necessário. São eles:

Operador	Ação
&	AND Lógico
|	OR Lógico
^	XOR (OR exclusivo)
~	NOT
>>	Shift Rigth
<<	Shift Left

Erros

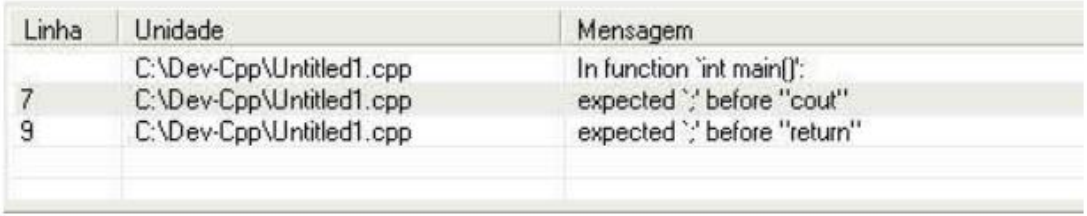
Verificando erros no código

- Quando compilamos um arquivo de código no Dev-C++, a janela indicadora do progresso da compilação é automaticamente aberta. Caso o arquivo de código não contenha nenhum erro, a compilação terminará e a janela de progresso permanecerá aberta para indicar que tudo correu bem (verifique o quadrado da janela chamado “status”: ele deverá indicar **Done** após o fim da compilação). Desta maneira, após o fim da compilação basta fechar a janela e executar o programa executável que foi gerado.

Erros

Verificando erros no código

- Caso nosso arquivo de código contenha uma ou mais linhas de códigos com erro, a compilação é interrompida para que estes erros (ou advertências) sejam verificados pelo programador. A janela de progresso da compilação é fechada, e a janela inferior do programa é maximizada mostrando todos os erros que foram encontrados durante a compilação do programa.



Linha	Unidade	Mensagem
	C:\Dev-Cpp\Untitled1.cpp	In function `int main()':
7	C:\Dev-Cpp\Untitled1.cpp	expected `;' before "cout"
9	C:\Dev-Cpp\Untitled1.cpp	expected `;' before "return"

- A figura acima mostra que a janela possui três colunas: linha, unidade e mensagem. A coluna linha indica a linha de código onde o erro foi encontrado; a coluna unidade indica o arquivo onde foi encontrado o erro e a coluna mensagem relata o tipo de erro encontrado. Um duplo clique em qualquer uma das indicações de erro nesta janela faz com que a linha de código onde o erro foi encontrado seja sublinhada em vermelho na janela de edição de código.

Debug

- Debug: serve para controlar o debug de um programa, que é a sua execução passo-a-passo

Aula 4 – Tipos de Dados

Professor Ricardo Demattê

IFSP – Unidade Salto

INSTITUTO FEDERAL
São Paulo
Câmpus Salto

Tipos primitivos de dados

- **Os principais tipos de dados utilizados em C e C++**

  - Um programa minimamente útil precisa manipular dados na memória do computador. A memória guarda apenas uma imensa sequência de bits e o que eles representam é indiferente para o funcionamento da máquina. Linguagens como C, entretanto, facilitam a vida do programador disponibilizando tipos de dados.
  - Um tipo de dado é um conjunto de possíveis valores e operações que podem ser efetuadas sobre esses valores. A linguagem C nos oferece alguns tipos de dados primitivos.
  - Existem 5 tipos de dados primitivos (pré-definidos) em C.

Tipo	Tamanho (em bits)	Intervalo
Char	8	-128 a 127
Int	16	-32768 a 32767
Float	32	3,4E-38 a 3,4E+38
double	64	1,7E-308 a 1,7E+308
void	0	sem valor

Tipos primitivos de dados

- A linguagem C tem definido os seguinte tipos primitivos:
- **char - int - float - double - void**

- **O tipo char**
- O tipo char é um tipo de informação que é capaz de armazenar um caractere alfanumérico. Geralmente, fazemos uso do tipo char para a construção de cadeias de caracteres, ou seja, um conjunto de letras que formam uma frase ou então, uma palavra qualquer.
- **O tipo int**
- O tipo int representa os números inteiros. Logo, podemos armazenar tanto números positivos como também, números negativos. Normalmente, o tipo int utiliza 2 bytes de memória e assim, é capaz de armazenar números que estejam no intervalo de -32768 até +32767.

Tipos primitivos de dados

- **O tipo float**
- O tipo float representa os números com ponto flutuante, os números que contenham casas decimais. Normalmente, o tipo float ocupa 4 bytes de memória.
- **O tipo double**
- O tipo "double" também representa os número com ponto flutuante, porém, conseguimos com o tipo double, muito mais precisão do que temos com o tipo "float". Isso porque, o tipo double faz uso de 8 bytes e assim, consegue armazenar números numa faixa muito extensa.
- **O tipo void**
- O tipo void representa um tipo sem tipo. Podemos pensar no tipo void como sendo a representação do vazio ou então do nada. Nós normalmente fazemos uso deste tipo em funções que não irão retornar valores. Logo, dissemos que a função retorna void, isto é, não retorna nenhum valor.

Tipos primitivos de dados

- **O tipo char**
- É um tipo de informação que é capaz de armazenar apenas um caractere alfanumérico.
- Representação: **%c**
- Exemplo:
- A variável **x** recebe o valor “s” e a variável **y** recebe o valor “n”

```c
#include <stdio.h>

int main()
{
  char x = 's';
  char y = 'n';
  printf("Resposta positiva = %c\n", x);
  printf("Resposta negativa = %c\n", y);
  return 0;
}
```

Tipos primitivos de dados

- **O tipo char [ ]**
- É um tipo de informação que é capaz de armazenar uma cadeia de caracteres alfanuméricos, ou uma **string**.
- Representação: **%s**
- Exemplo:
- A variável **nome** recebe o valor “John Doe”

```c
#include <stdio.h>

int main()
{
  char nome[50] = "John Doe";

  printf("Meu nome é %s.\n", nome);
  return 0;
}
```

Tipos primitivos de dados

- **O tipo int**
- O tipo int representa os números inteiros.
- Representação: **%d**
- Exemplo:
- A variável **x** recebe valor 5 e a variável **y** recebe valor 7
- A soma das duas variáveis resulta em **z**

```c
#include <stdio.h>

int main()
{
   int x=5;
   int y=7;
   int z;
   z = x+y;
   printf("A soma de %d + %d é igual a %d.\n",x,y,z);
   return 0;
}
```

Tipos primitivos de dados

- **O tipo float**
- O tipo float representa os números que contenham casas decimais.
- Representação: **%f ou %.2f (com duas casas decimais)**
- Exemplo: A variável **salario** recebe valor 1000.00
- Aplicar um aumento de 15% e mostrar o **novo_salario**

```c
#include <stdio.h>

int main()
{
    float salario = 1000.00;
    float novo_salario;

    novo_salario = salario + (salario*15/100);
    printf("Seu salario atual é R$ %.2f.\n",salario);
    printf("Seu novo salario será  R$ %.2f.\n",novo_salario);

    return 0;
}
```

Tipos primitivos de dados

- **O tipo double**
- O tipo "double" também representa os número com ponto flutuante, porém, conseguimos com o tipo double, muito mais precisão do que temos com o tipo "float“
- Representação: **%lf**
- Exemplo: O valor de **pi** é 3.14159265
- Calcular a área de uma circunferência de raio igual a 32.49 cm

```c
#include <stdio.h>

int main()
{
  double area;
  double pi=3.14159265;
  double r=32.49;

  area = pi*r*r;
  printf("A área é igual a %lf.\n",area);
  return 0;
}
```

Tipos primitivos de dados

- **O tipo double**
- O tipo "double" também representa os número com ponto flutuante, porém, conseguimos com o tipo double, muito mais precisão do que temos com o tipo "float“
- Representação: **%lf**
- Exemplo: O valor de **pi** é 3.14159265
- Calcular a área de uma circunferência de raio igual a 32.49 cm

```c
#include <stdio.h>

int main()
{
  double area;
  double pi=3.14159265;
  double r=32.49;

  area = pi*r*r;
  printf("A área é igual a %lf.\n",area);
  return 0;
}
```

Aula 5 – Funções de Entrada e Saída de Dados na Linguagem C

Professor Ricardo Demattê

IFSP – Unidade Salto

Função printf()

- A função **printf** é uma função de **saída de dados** da linguagem C
- Exemplo de **printf**

%d	inteiro
%f	float
%lf	double
%c	char
%s	palavra

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

int main()
{
    int num_inteiro = 5;
    float num_real = 49.98;
    double valor_pi = 3.14159265358979323846264338327950288419716939;
    char caracter = 'A';
    char nome[10] = "John";

    printf("O número inteiro: %d \n",num_inteiro);
    printf("O número real: %.2f \n",num_real);
    printf("O valor de pi é %.51f \n",valor_pi);
    printf("O caracter é %c \nO nome é %s.",caracter,nome);

    return(0);
}
```

Função scanf()

- A função **scanf** é uma função de **entrada de dados** da linguagem C
- Exemplo de **scanf**:

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

int main()
{
    int num_inteiro;
    float num_real;
    char caracter;
    char palavra[20];

    printf("Digite um caracter:\n");
    scanf("%c",&caracter);
    printf("Digite uma palavra:\n");
    scanf("%s",palavra);
    printf("Digite um número inteiro:\n");
    scanf("%d",&num_inteiro);
    printf("Digite um número real:\n");
    scanf("%f",&num_real);

    system ("clear");
    printf("O caracter é %c\nA palavra é %s \n",caracter,palavra);
    printf("O número inteiro: %d\nO número real: %f \n",num_inteiro,num_real);

    return(0);
}
```

Função scanf()

- Exemplo de utilização do **scanf**
  - Usando a biblioteca <stdio.h> e <stdlib.h>
  - Solicite um número inteiro (int) e armazene na variável “x”
  - Escreva o valor contido nesta variável “x”
  - O %d determina o tipo da variável a ser lida pela função scanf = inteira
  - O operador &x determina que é a leitura de uma variável “x”
  - O %d determina o tipo da variável a ser escrita pela função printf = inteira

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

int main()
{
    int x;
    printf("Digite um número qualquer:\n");
    scanf("%d",&x);
    printf("O número digitado foi: %d.",x);
    return 0;
}
```

%d	inteiro
%f	float
%lf	double
%c	char
%s	palavra

Função scanf()

- Exemplo de utilização do **scanf**
  - Usando a biblioteca <stdio.h> e <stdlib.h>
  - Solicite um número real (float) e armazene na variável “salario”
  - Escreva o valor contido nesta variável “salario”
  - O %f determina o tipo da variável a ser lida pela função scanf = real
  - O operador &salario determina que é a leitura de uma variável “salario”
  - O %f determina o tipo da variável a ser escrita pela função printf = inteira

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

int main()
{
    float salario;

    printf("Digite seu salário:\n");
    scanf("%f",&salario);
    printf("Seu salário é R$ %.2f.",salario);
    return 0;
}
```

%d	inteiro
%f	float
%lf	double
%c	char
%s	palavra

Função scanf() para strings

- Exemplo de utilização do **scanf**
  - Usando a biblioteca <stdio.h> e <stdlib.h>
  - Solicite um nome tipo caractere (string) e armazene na variável “codigo”
  - O valor inserido, neste caso, deve ser uma palavra simples. Ex: AEP
  - Neste caso, uma palavra simples não pode conter espaços em branco.
  - Escreva o valor contido nesta variável “codigo”

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

int main()
{
    char codigo[10];
    printf("Digite seu código:\n");
    scanf("%s",&codigo);
    printf("O seu código é %s.",codigo);
    return 0;
}
```

%d	inteiro
%f	float
%lf	double
%c	char
%s	palavra

Função gets()

- Exemplo de utilização do gets no lugar do **scanf**
  - Usando a biblioteca <stdio.h> e <stdlib.h>
  - Solicite um nome composto(string), idade(int) e altura(float) de uma pessoa.
  - Depois de coletar estes dados, exiba todas as informações em uma linha.

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

int main()
{
    char nome[50];
    int idade;
    float altura;

    printf("Digite seu nome:\n");
    gets(nome);
    printf("%s, digite sua idade:\n",nome);
    scanf("%d",&idade);
    printf("%s, digite sua altura:\n",nome);
    scanf("%f",&altura);

    printf("%s tem %d anos e sua altura é %.2fm.",nome,idade,altura);
    return 0;
}
```

%d	inteiro
%f	float
%lf	double
%c	char
%s	palavra

Exercícios

- **Exercício 1**
  - Escreva a seguinte frase na tela “Instituto Federal de São Paulo”.
- **Exercício 2**
  - Pergunte seu e-mail, armazene em uma variável chamada “email”.
  - Ao final escreva o valor contido na variável “email”.
- **Exercício 3**
  - Pergunte o primeiro nome, o último sobrenome, e-mail, RG, CPF e renda familiar de uma pessoa. Ao final mostre todos estes dados em apenas uma linha.
- **Exercício 4**
  - Pergunte nome, matricula e o salário bruto de um funcionário.
  - O salário liquido equivale ao salário bruto menos os impostos (27%).
  - Ao final escreva o nome, matrícula e o salário liquido deste funcionário.
- **Exercício 5**
  - Pergunte nome, prontuário e 4 notas de um aluno.
  - Ao final escreva o nome, prontuário e a média deste aluno.

Exercícios

- **Exercício 6**
  - Informar um preço de um produto e calcular novo preço com desconto de 7,5%.
- **Exercício 7**
  - Leia uma distancia em milhas e apresente-a convertida em quilômetros. A fórmula de conversão é **K=1,61∗M**, sendo K a distancia em quilômetros e M em milhas.
- **Exercício 8**
  - Leia um valor de comprimento em polegadas e apresente-o convertido em centímetros. A formula de conversão é **C=P∗2,54** sendo C o comprimento em centímetros e P o comprimento em polegadas..
- **Exercício 9**
  - Leia um valor inteiro qualquer em segundos e mostre-o em quantidade equivalente em horas, minutos e segundos.
- **Exercício 10**
  - Uma empresa contrata um encanador a R$ 30,00 por dia. Faça um programa que solicite o numero de dias trabalhados pelo encanador e imprima a quantia líquida que devera será paga, sabendo-se que são descontados 5,25% do Imposto Sobre Serviços (ISS) da Prefeitura.

A biblioteca <math.h>

Funções Matemáticas

Função	Exemplo	Comentário
`ceil`	`ceil(x)`	Arredonda o número real para cima; ceil(3.2) é 4
`cos`	`cos(x)`	Cosseno de x (x em radianos)
`exp`	`exp(x)`	**e** elevado à potencia x
`fabs`	`fabs(x)`	Valor absoluto de x
`floor`	`floor(x)`	Arredonda o número deal para baixo; floor(3.2) é 3
`log`	`log(x)`	Logaritmo natural de x
`log10`	`log10(x)`	Logaritmo decimal de x
`pow`	`pow(x, y)`	Calcula x elevado à potência y
`sin`	`sin(x)`	Seno de x
`sqrt`	`sqrt(x)`	Raiz quadrada de x
`tan`	`tan(x)`	Tangente de x

```
#include <math.h>
```

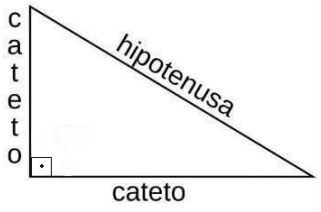
Usando a biblioteca <math.h>

- Exemplo de utilização do **scanf** e **gets**
  - Usando a biblioteca <stdio.h> e <stdlib.h> e <math.h>
  - Solicite o valor dos catetos de um triangulo retângulo e calcule o valor da hipotenusa. Mostre o resultado com 2 casas decimais.

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

int main()
{
    float cat_a,cat_b,hipo;

    printf("Digite o valor do cateto a:\n");
    scanf("%f",&cat_a);
    printf("Digite o valor do cateto b:\n");
    scanf("%f",&cat_b);
    hipo = sqrt((pow(cat_a,2))+(pow(cat_b,2)));
    printf("A hipotenusa é %.2f.",hipo);
    return 0;
}
```



%d	inteiro
%f	float
%lf	double
%c	char
%s	palavra

Exercícios – Use a biblioteca <math.h>

- **Exercício 11**
  - Elabore um algoritmo que calcule e mostre a raiz quadrada de um número inteiro. Mostre o novo valor da nota com uma casa decimal.
- **Exercício 12**
  - Elabore um algoritmo que leia dois números reais.
  - Imprima a soma, subtração, multiplicação entre estes números. Calcule também a divisão por 10 e a potencia cubica destes dois números. Apresente os números com 2 casas decimais.
- **Exercício 13**
  - Dado um ângulo qualquer, determine o seno, cosseno e tangente deste ângulo. Mostre o resultado com quatro casas decimais.
- **Exercício 14**
  - Dado uma nota qualquer, com duas casas decimais, arredonde para cima e mostre o novo valor da nota arredondada.
- **Exercício 15**
  - Fazer um programa em C que solicite um valor em metros e mostre o correspondente em decímetros, centímetros e milímetros.

Aula 6 – Funções de Entrada e Saída de Dados na Linguagem C++

Professor Ricardo Demattê

IFSP – Unidade Salto

Estudo dos comandos cout e cin

CIN e COUT

- A linguagem C++ possui uma ótima biblioteca de classes relacionadas ao controle de entrada e saídas de dados.
- Como você deve ter percebido, a classe **cout** serve para exibir valores - seja o valor de uma variável ou uma frase – enquanto que **cin** serve para armazenar valores recebidos através do teclado em variáveis.
- Tínhamos na linguagem C as funções printf e scanf para executar estas mesmas funções. Na verdade, printf e scanf também estão presentes em C++ (assim como todas as funções padrões de C), e podemos utilizá-las caso desejemos.
- Porém, os comandos cin e cout facilitam muito a vida do programador, por serem mais “inteligentes” que printf e scanf.

Utilização de cout()

- **Uso do operador de inserção.**
  - Utilizamos cout em conjunto com o operador de inserção **<<**
  - O operador **<<** indica ao comando cout que um dado deve ser exibido na tela, além de identificar automaticamente qual o tipo deste dado e como ele deve ser formatado para exibição na tela.
  - Assim, não precisamos informar à cout que estamos enviando um inteiro, um real ou uma string, como fazíamos em C: o operador **<<** se encarrega desta identificação, bastando para o operador indicar o nome da variável.

- **Utilizamos o cout para exibir valores na tela.**
- **A sintaxe utilizada é:**
  - **cout << <valor, string, variável, ponteiro, etc>;**

Exemplos do uso de cout()

- Exemplo: mostrando uma mensagem de texto na tela

  - Use a biblioteca **<iostream>** que é a biblioteca padrão do C++
  - Esta biblioteca possui os comandos básicos do C++ e é utilizada na maioria dos programas feitos em C++
  - Usando nomenclatura padrão C++ : “using namespace std;”
  - Coloque esta mensagem na tela : “IFSP - Campus Salto”
  - Utilize o comando de saída **COUT**

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      cout<<"IFSP - Campus Salto";
7      return 0;
8  }
```

Exemplos do uso de cout()

- Exemplo: mostrando uma mensagem de texto e o valor de uma variável na tela

  - Use a biblioteca <iostream>
  - Atribua o valor 10 na variável “x”
  - Exiba o valor da variável “x”
  - Utilize o comando COUT

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int x=10;
7      cout<<"O valor de x é "<<x;
8      return 0;
9  }
```

Utilização de cin()

- Utilizamos o comando **cin** para obter ou ler valores digitados pelo usuário através do teclado.
- A sintaxe utilizada é a seguinte:
- **cin >> variavel_destino;**
- Exemplo: digite e mostre um número usando CIN e COUT

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int num;
7      cout<<"Digite um número:\n";
8      cin>>num;
9      cout<<"O número digitado foi "<<num;
10     return 0;
11 }
```

Utilização de cin() com strings

- Faça um programa que leia o **primeiro nome** de uma pessoa e mostre uma mensagem de boas vindas
- Neste caso, utilize um **nome simples**, que não tenha espaços em branco
- Use CIN e COUT e a biblioteca <string>

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      char nome[20];
7
8      cout<<"Digite seu primeiro nome:\n";
9      cin>>nome;
10
11     system ("clear");
12     cout<<"Benvindo(a), "<<nome;
13
14     return(0);
15 }
```

A biblioteca <string> e getline()

- Faça um programa que leia o **nome completo**, telefone e e-mail de uma pessoa e mostre estes dados em linhas distintas.
- Neste caso, utilize um **nome composto**, que contenha espaços em branco.
- Use a biblioteca **<string>,** e o comando **getline**.

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main()
6  {
7      string nome,email,fone;
8      cout<<"Digite seu nome completo:\n";
9      getline (cin,nome);
10     cout<<"Digite seu e-mail:\n";
11     getline (cin,email);
12     cout<<"Digite seu telefone:\n";
13     getline (cin,fone);
14     cout<<"Bem vindo ao IFSP, "<<nome<<"!!!\n";
15     cout<<"Seu telefone é "<<fone<<"\n";
16     cout<<"Seu e-mail é "<<email<<"\n";
17     return 0;
18 }
```

Utilização de cin() com números

- Faça um programa que leia a altura de 3 pessoas e calcule a média.
- Utilize variáveis tipo **float** e **double** e mostre a altura com 2 casas decimais

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      float alt1,alt2,alt3;
7      double media=0;
8
9      cout<<"Digite a altura da 1ª pessoa:\n";
10     cin>>alt1;
11     cout<<"Digite a altura da 2ª pessoa:\n";
12     cin>>alt2;
13     cout<<"Digite a altura da 3ª pessoa:\n";
14     cin>>alt3;
15
16     media=(alt1+alt2+alt3)/3;
17     cout.precision (3);
18     cout<<"A média das alturas é "<<media<<"m.\n";
19     return 0;
20 }
```

A biblioteca <iomanip>

- A biblioteca <iomanip> é utilizada arredondamento de uma variável e outros tipos de manipulação de dados
  - Crie um algoritmo que leia duas notas de um aluno, calcule a média.
  - Apresente o resultado com duas casas decimais.
  - Use a biblioteca <iomanip>

```
1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4
5  int main ()
6  {
7      float nota1,nota2,media;
8      cout << "Digite sua a nota 1:\n";
9      cin >> nota1;
10     cout << "Digite sua a nota 1:\n";
11     cin >> nota2;
12     media = (nota1+nota2)/2;
13     cout << "A média obtida foi "<<fixed<<setprecision(2)<<media;
14     return 0;
15 }
```

A biblioteca <cmath>

- A biblioteca <cmath> é similar a biblioteca <math.h
  - Crie um algoritmo que leia um número inteiro NUM e mostre o seu dobro, triplo e a raiz quadrada. Apresente cada resultado em uma linha distinta.
  - Use CIN e COUT e a biblioteca **<cmath>** usada no C++

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4
5  int main()
6  {
7      int num,dobro,triplo;
8      float raiz;
9      cout<<"Digite um número qualquer:\n";
10     cin>>num;
11     dobro = 2*num;
12     triplo = 3*num;
13     raiz = sqrt(num);
14     cout<<"O dobro de "<<num<<" é "<<dobro<<"\n";
15     cout<<"O triplo de "<<num<<" é "<<triplo<<"\n";
16     cout<<"A raiz quadrada de "<<num<<" é "<<raiz<<"\n";
17     return 0;
18 }
```

Exercícios

- **Exercício 16**
  - Escreva o seu nome completo na tela.
- **Exercício 17**
  - Escreva o nome de todos os 8 planetas do sistema solar.
  - Pule uma linha toda vez que exibir um planeta.
- **Exercício 18**
  - Faça um programa que leia o nome de uma pessoa e mostre uma mensagem de boas vindas.
- **Exercício 19**
  - Crie um algoritmo que leia um número inteiro qualquer.
  - Mostre o seu dobro, triplo, o cubo e a raiz quadrada.
- **Exercício 20**
  - Escreva um programa que pergunte a quantidade de Km percorridos por um carro alugado e a quantidade de dias pelos quais ele foi alugado.
  - Calcule o preço a pagar, sabendo que o carro custa R$ 60 por dia e R$ 0,15 por Km rodado.

Exercícios

- **Exercício 21**
  - Faça um programa que leia o comprimento do cateto oposto e do cateto adjacente de um triângulo.
  - Calcule o mostre o comprimento da hipotenusa.
- **Exercício 22**
  - Dado uma nota qualquer, com duas casas decimais, arredonde para cima
  - Mostre o novo valor da nota com uma casa decimal.
- **Exercício 23**
  - Fazer um programa em C++ que solicite um valor em metros e mostre o correspondente em decímetros, centímetros e milímetros.
- **Exercício 24**
  - Desenvolva um programa que leia as duas notas de um aluno, calcule e mostre a sua média. Apresente com uma casa decimal.
- **Exercício 25**
  - Faça um programa que leia um número inteiro e mostre na tela o seu sucessor e seu antecessor.

Exercícios

- **Exercício 26**
  - Black Friday: faça um programa que leia o preço de um produto e calcule o seu novo preço, com 35% de desconto. O custo do frete é de 7,5% do valor do produto. O valor do Imposto sobre Produtos Industrializados (IPI) é de 18%. Calcule o valor total a ser pago por este produto, em reais.
- **Exercício 27**
  - A extensão do circuito de Interlagos é de 4.309 metros. Ayrton Senna no GP do Brasil em 1991 cravou a *pole position* com 1min16s. Calcule a velocidade média que Ayrton Senna fez nesta volta. Apresente o resultado em km/h.
- **Exercício 28**
  - Fazer um programa que calcule quanto tempo levaria para uma pessoa que ganha salário mínimo ficar milionário, sem gastar nada. Apresente o resultado em anos.
- **Exercício 29**
  - O raio do planeta Terra é de 6371 km. Calcule a sua circunferência, em km.
- **Exercício 30**
  - Sabendo que a distância entre o Sol e a Terra é de 149.600.000 km e a velocidade da luz é de 299.792.458 metros por segundo, calcule quanto o tempo a luz leva para percorrer o caminho entre o Sol e a Terra. Apresente o resultado em minutos.

Aula 7 – Estrutura Condicional

Professor Ricardo Demattê

IFSP – Unidade Salto

Estrutura condicional

- **Estrutura condicionais**
  - São comandos utilizados em uma linguagem de programação para determinar qual a ordem e quais comandos devem ser executados pelo programa em uma dada condição.
  - Geralmente, estas estruturas utilizam expressões condicionais.
- **A declaração IF**
  - Utilizamos a declaração if quando desejamos que o programa teste uma ou mais condições e execute um ou outro comando de acordo com o resultado deste teste.
  - A sintaxe de if é a seguinte:

```
1  if (condição)
2  {
3      comandos;
4  }
```

Estrutura condicional: if e else

- **IF e ELSE**
  - A declaração if testará a condição expressa entre parênteses.
  - Caso a condição seja verdadeira, os comandos declarados entre as chaves serão executados. As chaves não são obrigatórias.
  - A declaração else é opcional: podemos utilizá-la para determinar um conjunto de comandos que serão executados caso a condição testada seja falsa.
  - Note que somente um dos conjuntos de comandos será executado, nunca os dois: caso a condição seja verdadeira, o bloco pertencente a if será executado; caso a condição falhe, o bloco pertencente a else será executado.

```
1  if (condição)
2  {
3      comandos;
4  }
5  else
6  {
7      comandos;
8  }
```

Estrutura condicional: IF

- **Exemplo – Usando IF**
  - O programa abaixo ilustra o uso da declaração **IF**, obtendo um número do usuário e verificando se este valor é maior, menor ou igual a 50.

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int num;
7      cout<<"Digite um número qualquer:\n";
8      cin>>num;
9      if (num==50)
10     {
11         cout<<"O número "<<num<<" digitado é igual a 50.";
12     }
13     if (num<50)
14     {
15         cout<<"O número "<<num<<" digitado é menor que 50.";
16     }
17     if (num>50)
18     {
19         cout<<"O número "<<num<<" digitado é maior que 50.";
20     }
21     return 0;
22 }
```

Estrutura condicional: IF e ELSE

- **Exemplo – Usando IF e ELSE**
  - O programa abaixo ilustra de maneira simples o uso da declaração **if-else**, obtendo um número do usuário e verificando se este valor é par ou impar.

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int num;
7      cout<<"Digite um número qualquer:\n";
8      cin>>num;
9      if (num%2==0)
10     {
11         cout<<"O número "<<num<<" digitado é par.";
12     }
13     else
14     {
15         cout<<"O número "<<num<<" digitado é impar.";
16     }
17     return 0;
18 }
```

Estrutura condicional: IF e ELSE

- O programa abaixo verifica se o ano atual é bissexto, utilizando os operadores lógicos **E** e **OU** de acordo com a definição de ano bissexto.

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int ano;
7
8      cout<<"Insira o ano para verificar se é bissexto:\n";
9      cin>>ano;
10
11     system("clear");
12     if (ano%4==0 && ano%100!=0 || ano%400==0)
13     {
14         cout<<"O ano "<<ano<<" é bissexto.";
15     }
16     else
17     {
18         cout<<"O ano "<<ano<<" não é bissexto.";
19     }
20
21     return(0);
22 }
```

IF aninhado

- **A linguagem C permite que se façam IFs aninhados**
  - O if aninhado é simplesmente um **IF** dentro da declaração de um outro **IF** externo.
  - O único cuidado que devemos ter é o de saber exatamente a qual **IF** um determinado else está ligado.
  - Esta funcionalidade é útil quando precisamos testar várias condições, uma dentro da outra.
- **Exemplo**

1. Faça um programa que solicite um número qualquer.
2. Se este número for igual a 10, escreva a mensagem:
   - " Você acertou, o número é 10"
3. Senão verifique se o número é maior ou menor que 10 e escreva uma mensagem correspondente para cada cenário:
   - " O número é menor que 10"
   - " O número é maior que 10"

IF aninhado

- **Exemplo de um IF aninhado**

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int num;
7      cout<<"Digite um número qualquer:\n";
8      cin>>num;
9      if (num==10)
10     {
11         cout<<"Você acertou !!\n";
12         cout<<"O número "<<num<<" digitado é 10.";
13     }
14     else
15     {
16         if (num>10)
17         {
18             cout<<"O número "<<num<<" digitado é maior que 10.";
19         }
20         else
21         {
22             cout<<"O número "<<num<<" digitado é menor que 10.";
23         }
24     }
25     return 0;
26 }
```

Exercícios

- **Exercício 31**
  - Ler um valor inteiro e escrever se é positivo ou negativo
  - Se digitar zero, escreva a mensagem “valor nulo”.
- **Exercício 32**
  - Ler dois valores (considere que não serão lidos valores iguais)
  - Escrever o maior deles
  - Escrevê-los em ordem crescente.
- **Exercício 33**
  - As maçãs custam R$ 1,30 cada se forem compradas menos de uma dúzia, e a partir de 12, as mesmas maçãs custam R$ 1,00 cada.
  - Escreva um programa que leia o número de maçãs compradas, calcule e escreva o custo total da compra.
- **Exercício 34**
  - Elabore um programa que solicite o nome do usuário e que este escolha o idioma de sua preferencia: inglês, espanhol ou português.
  - Emita uma mensagem de boas vindas personalizada para este usuário, de acordo com o idioma escolhido.

Exercícios

- **Exercício 35**
  - Ler o nome de 2 times e o número de gols marcados na partida, para cada time.
  - Escrever o nome do vencedor.
  - Caso não haja vencedor deverá ser impressa a palavra “empate”.
- **Exercício 36**
  - Ler o ano atual e o ano de nascimento de uma pessoa.
  - Escrever uma mensagem que diga se ela poderá ou não votar este ano (não é necessário considerar o mês em que a pessoa nasceu).
- **Exercício 37**
  - Escreva um programa que leia a velocidade de um carro.
  - Se ele ultrapassar 80 km/h, mostre uma mensagem dizendo que ele foi multado.
  - A multa vai custar R$ 7,00 por cada Km acima do limite.
- **Exercício 38**
  - Desenvolva um programa que pergunte a distância de uma viagem em km.
  - Calcule o preço da passagem, cobrando R$ 0,50 por km para viagens de até 200km e R$ 0,45 para viagens mais longas.

Exercícios

- **Exercício 39**
  - Faça um algoritmo para ler um número que é um código de usuário. Caso este código seja diferente de um código armazenado internamente no algoritmo (igual a 1234) deve ser apresentada a mensagem 'Usuário inválido!'.
  - Caso o Código seja correto, deve ser lido outro valor que é a senha. Se esta senha estiver incorreta (a certa é 9999) deve ser mostrada a mensagem 'senha incorreta'. Caso a senha esteja correta, deve ser mostrada a mensagem 'Acesso permitido'.
- **Exercício 40**
  - Uma fruteira está vendendo frutas com a seguinte tabela de preços:

	Até 5 Kg	**Acima de 5 Kg**
Morango	R$ 2,50 por Kg	R$ 2,20 por Kg
Maçã	R$ 1,80 por Kg	R$ 1,50 por Kg

  - Escreva um algoritmo para ler a quantidade (em Kg) de morangos e a quantidade (em Kg) de maças adquiridas e escreva o valor a ser pago pelo cliente.

Exercícios

- **Exercício 41**
  - Faça um programa que leia um número inteiro e verifique se ele é ímpar e múltiplo de 5. Mostre na tela este número, caso contrário escreva “número inválido”.
- **Exercício 42**
  - Escreva um programa que pergunte o nome e o salário de um funcionário e calcule o valor do seu aumento.
  - Para salários superiores a R$ 1.250,00, calcule um aumento de 10%. Para os inferiores ou iguais, o aumento é de 15%.
  - Mostre o nome do funcionário, o salário atual e o salário com aumento.
- **Exercício 43**
  - A jornada de trabalho em uma empresa é de 40hs semanais ou 160hs mensais.
  - O funcionário que trabalhar mais de 160 horas mensais receberá hora extra, cujo cálculo é o valor da hora regular com um acréscimo de 50%. O valor da hora trabalhada nesta empresa para todos os funcionários é de R$15,62.
  - Faça um programa que leia o nome do funcionário e o número de horas trabalhadas por ele em um mês. Mostre o nome e o salário total do funcionário, que deverá ser acrescido das horas extras (se houver).
  - Obs: O funcionário que trabalhar menos que 150 horas mensais deve ser demitido.

Exercícios

- **Exercício 44**
  - Faça um programa que solicite 2 números quaisquer.
  - Restrição: Se algum dos números digitados for igual a zero, mostre a mensagem “Zero é número inválido” e solicite um novo número.
  - Faça operações matemáticas de acordo com o que for escolhido pelo usuário em um menu de opções de uma calculadora:

  - Opção 1 – Soma
  - Opção 2 – Subtração
  - Opção 3 – Multiplicação
  - Opção 4 – Divisão
  - Opção 5 – Potenciação

  - Mostre os números digitados pelo usuário
  - A operação matemática realizada
  - O resultado da operação

Exercícios

Exercício 45

- O imposto de renda é calculado com base no salário bruto que um determinado funcionário recebe.
- A tabela do imposto serve de base de calculo do imposto.

Base de cálculo	Alíquota
de 0,00 até 1.903,98	isento
de 1.903,99 até 2.826,65	7,50%
de 2.826,66 até 3.751,05	15,00%
de 3.751,06 até 4.664,68	22,50%
a partir de 4.664,68	27,50%

Receita Federal

- Desenvolva um sistema que receba o nome do funcionário, sua matrícula funcional e o salário sem descontos (salário bruto).
- Apresente um relatório com o nome do funcionário, sua matrícula e o salário líquido (já com os descontos do imposto de renda.

Estrutura condicional: ELSE IF

- **IF ELSE IF**
  - A estrutura if-else-if é apenas uma extensão da estrutura if-else.
  - Sua forma geral pode ser escrita como sendo:

```
if     (condição_1)  declaração_1;
else if (condição_2)  declaração_2;
else if (condição_3)  declaração_3;
...
else if (condição_n) declaração_n;
else declaração_default;
```

  - A estrutura acima funciona da seguinte maneira: o programa começa a testar as condições começando pela 1 e continua a testar até que ele ache uma expressão cujo resultado dê diferente de zero (falso).
  - Neste caso ele executa a declaração correspondente (verdadeiro).
  - Só uma declaração será executada, ou seja, só será executada a declaração equivalente à primeira condição que der diferente de zero (falso).
  - A última declaração (default) é a que será executada no caso de todas as condições darem zero e é opcional.

Exemplo

- **Emitir uma mensagem de acordo com o horário:**
- **Bom dia, boa tarde ou boa noite**

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int hora = 22;
7      if (hora < 12)
8      {
9          cout << "Bom dia.";
10     }
11     else if (hora < 18)
12     {
13         cout << "Boa tarde.";
14     }
15     else
16     {
17         cout << "Boa noite.";
18     }
19     // Saída "Boa noite."
20     return 0;
21 }
```

Exercícios

- **Exemplo:**
  - Dado um número qualquer, verifique se este número:
  - É maior, menor ou igual a 10

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int num;
7      cout<<"Digite um número:\n";
8      cin>>num;
9      system ("clear");
10
11     if (num==10)
12     {
13         cout<<"Você digitou 10";
14     }
15     else if (num>10)
16     {
17         cout<<"Você digitou um número maior que 10";
18     }
19     else if (num<10)
20     {
21         cout<<"Você digitou um número menor que 10";
22     }
23     return 0;
24 }
```

Exercícios

- **Exemplo:**
  - Dado dois números quaisquer, verifique se os números são iguais:
  - Se forem diferentes, identifique quem é o maior

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

int main()
{
    int n1,n2;
    cout<<"Digite um número:\n";
    cin>>n1;
    cout<<"Digite outro número:\n";
    cin>>n2;
    system("clear");
    if (n1==n2)
    {
        cout<<"Os números são iguais";
    }
    else if (n1>n2)
    {
        cout<<"N1 é maior que N2";
    }
    else if (n2>n1)
    {
        cout<<"N2 é maior que N1";
    }
    return 0;
}
```

Exercícios

- **Exercício 46**
  - Faça um algoritmo que receba um número e mostre uma mensagem caso este número seja maior que 120, menor que 60 ou igual a 80. Exibir resultado.
- **Exercício 47**
  - Faça um algoritmo que leia os valores A, B, C e diga se a soma de A + B é maior ou menor que C. Mostre uma mensagem com o resultado.
- **Exercício 48**
  - Ler 3 valores (considere que não serão informados valores iguais) e exibir a soma dos 2 maiores e a multiplicação dos 2 menores
- **Exercício 49**
  - Leia um número N qualquer e verifique se ele é impar e múltiplo de 3. Mostrar somente números que atendam este critério, senão emitir mensagem de erro.
- **Exercício 50**
  - Dado a velocidade medida de um carro que passa por um radar móvel na rodovia, determine o tipo de multa:
  - Abaixo de 110km/h (sem multa), de 111km/h até 125km/h (multa leve), de 126km/h até 140km/h (multa grave) e acima de 140km/h (multa gravíssima)

Exercícios

- **Exercício 51**
  - Escrever um algoritmo que leia 3 valores para A,B e C . Verifique se os valores (A,B e C) são iguais entre si, e mostre esta informação. Porém se forem diferentes, mostre-os em ordem crescente.
- **Exercício 52**
  - Escrever um algoritmo que leia os dados de uma pessoa (nome, sexo, idade, altura e saúde) e informe se está apta ou não para cumprir o serviço militar obrigatório. Uma pessoa apta deve ser do sexo masculino, ter 18 anos, 1.70 de altura e estar em com boa saúde.
- **Exercício 53**
  - Elabore um algoritmo que, dada a idade de um nadador. Classifique-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
  - Sênior = 18 - 25 anos

  - Apresentar mensagem “idade fora da faixa etária” quando for outro ano não contemplado.

Exercícios

- **Exercício 54**
  - O QI (quociente de inteligência) médio do brasileiro é 88.
  - Informe o QI de uma pessoa e verifique se esta **acima**, **igual** ou **abaixo** da média.
- **Exercício 55**
  - O cálculo do **peso ideal** foi desenvolvido pela Organização Mundial de Saúde (OMS) e é feito utilizando a fórmula do Índice de Massa Corporal:
  - **IMC = peso / altura²**
  - Faça um algoritmo que solicite o nome, peso e altura de uma pessoa.

1. Exiba o nome e a classificação de peso, de acordo com a tabela abaixo.
2. Também exiba o peso ideal para esta pessoa: (considere IMC ideal = **22**)

Abaixo do peso	Abaixo de 18.4
Peso normal	**18.5 - 24.9**
Sobrepeso	25.0 - 29.9
Obesidade Grau I	30.0 - 34.9
Obesidade Grau II	35.0 - 39.9
Obesidade Grau III	Acima de 40

Estrutura condicional: Switch

SWITCH, CASE BREAK

- A declaração switch é uma maneira fácil e elegante de se fazer uma tomada de decisão com múltiplas escolhas.
- Na declaração switch, a variável é sucessivamente testada contra uma lista de inteiros ou constantes caractere. Quando uma associação é encontrada, o conjunto de comandos associado com a constante é executado.
- Podemos fazer uma analogia entre o switch e a estrutura if-else-if apresentada anteriormente. A diferença fundamental é que a estrutura switch não aceita expressões. Aceita apenas constantes.
- O switch testa a variável e executa a declaração cujo case corresponda ao valor atual da variável.
- A declaração default é opcional e será executada apenas se a variável, que está sendo testada, não for igual a nenhuma das constantes.
- O comando break, faz com que o switch seja interrompido assim que uma das declarações seja executada.
- Mas ele não é essencial ao comando switch. Se após a execução da declaração não houver um break, o programa continuará executando.

Exemplo: Switch

- **Escolher um dia da semana entre 1, 2, 3, 4, 5, 6 ou 7**

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

int main()
{
    int dia = 4;
    switch (dia)
    {
        case 1:
            cout << "Segunda-feira";
        break;
        case 2:
            cout << "Terça-feira";
        break;
        case 3:
            cout << "Quarta-feira";
        break;
        case 4:
            cout << "Quinta-feira";
        break;
        case 5:
            cout << "Sexta-feira";
        break;
        case 6:
            cout << "Sabado";
        break;
        case 7:
            cout << "Domingo";
        break;
    }
// Saida "Quinta-feira" (dia 4)
    return 0;
}
```

Exemplo: Switch

- **Exemplo utilizando switch**
  - Verificar e escrever mensagem se um número é igual a 1, 2 ou 3
  - Escrever mensagem de erro se for diferente

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int num;
7      cout<<"Digite 1, 2 ou 3\n";
8      cin>>num;
9      system ("clear");
10
11     switch (num)
12     {
13         case 1:
14             cout<<"Você digitou 1";
15             break;
16         case 2:
17             cout<<"Você digitou 2";
18             break;
19         case 3:
20             cout<<"Você digitou 3";
21             break;
22         default:
23             cout<<"Você digitou um número inválido...";
24             break;
25     }
26     return 0;
27 }
```

Exemplo: Switch

- **Digitando a idade, saiba qual é a fase da sua vida**

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

main()
{
    int idade;
    cout<<"Digite a idade de uma pessoa:\n";
    cin>>idade;
    switch (idade)
    {
        case 0 ... 3:
            cout<<"Idade: "<<idade;
            cout<<"\nFase da vida: Primeira infância.";
        break;
        case 4 ... 6:
            cout<<"Idade: "<<idade;
            cout<<"\nFase da vida: Pré escolar.";
        break;
        case 7 ... 10:
            cout<<"Idade: "<<idade;
            cout<<"\nFase da vida: Segunda infância.";
        break;
        case 11 ... 17:
            cout<<"Idade: "<<idade;
            cout<<"\nFase da vida: Adolescência.";
        break;
        case 18 ... 65:
            cout<<"Idade: "<<idade;
            cout<<"\nFase da vida: Idade Adulta.";
        break;
        case 65 ... 150:
            cout<<"Idade: "<<idade;
            cout<<"\nFase da vida: Velhice.";
        break;
        default:
            cout<<"Idade inválida !!!";
    }
    return 0;
}
```

Exercícios

Exercício 56

- Um carro possui 5 marchas e a marcha ré.
- Digite um número de 0 a 5, sendo 0 a marcha ré.
- De acordo com o número digitado, mostre a marcha utilizada.
- Se o número digitado estiver fora do limite entre 0 e 5, deve ser exibida uma mensagem de erro: “Marcha inválida”
- Use a estrutura **switch-case** neste exercício

Exercício 57

- Uma máquina executa 5 operações, de acordo com os códigos abaixo:
- 1001 = Ligar, 1002 = Checagem de Funcionamento
- 1003 = Operação Padrão, 1004 = Resfriamento,
- 1005 = Desligar
- De acordo com o número digitado, mostre a operação utilizada.
- Se o número digitado for diferente dos códigos fornecidos, deve ser exibida uma mensagem de erro: “Código inválido.”
- Use a estrutura **switch-case** neste exercício

Exercícios

- **Exercício 58**
  - Use a estrutura **switch-case** neste exercício
  - Digite um número de 1 a 12, de acordo com os meses do ano.
  - De acordo com o número digitado, mostre a descrição do mês por extenso
  - Se o número digitado estiver fora do limite entre 1 e 12, deve ser exibida uma mensagem de erro.

Exercícios

- **Exercício 59**
  - Use a estrutura **switch-case** neste exercício
  - Elabore um algoritmo que dado o peso de um lutador de boxe classifique-o em uma das seguintes categorias:
  - **mosca-ligeiro** (até 49kg)
  - **mosca** (até 52kg)
  - **galo** (até 56kg)
  - **leve** (até 60kg)
  - **médio-ligeiro** (até 64kg)
  - **meio-médio** (até 69kg)
  - **médio** (até 75kg)
  - **meio-pesado** (até 81kg)
  - **pesado** (até 91kg)

  - **superpesado** (acima de 91kg até 130kg).
  - Apresentar mensagem "**peso fora da faixa**" quando peso for maior que 130kg.

Exercícios

- **Exercício 60**
  - Use a estrutura **switch-case** neste exercício
  - Elabore um algoritmo que leia a nota do ENEM obtida por um estudante e verifique se o mesmo foi aprovado no vestibular da USP.
  - Se o aluno foi aprovado, mostre quais cursos ele está apto a cursar:
  - **Gestão Ambiental** (643) **Turismo** (651)
  - Geologia (658) **Educação Física** (744)
  - **Jornalismo** (796) **Engenharia Civil** (801)
  - **Relações Internacionais** (809)
  - **Engenharia Mecânica** (812) **Física** (826)
  - **Engenharia da Computação** (849)
  - Apresentar mensagem "**Reprovado na USP**" quando nota for menor que 643, que é a nota de corte do ENEM para a USP.

Aula 8 – Estruturas de Repetição

Professor Ricardo Demattê

IFSP – Unidade Salto

Estruturas de repetição: for

- **A declaração FOR**
  - Utiliza-se a declaração **for** para realizar tarefas repetitivas dentro de um programa, como somar todos os elementos de uma matriz ou exibir na tela uma sequência grande de valores.
  - A declaração for tem o seguinte formato:

```
for (valor_inicial; condição_testada; valor incremento)
{
    comandos;
}
```

  - A declaração **for** é o que chamamos de laço ou loop em programação: um conjunto de comandos que serão executados repetidamente até que uma determinada condição falhe e termine o laço.
  - Em **for**, determinamos o número de repetições desejadas através de uma variável de controle que será modificada pelos argumentos da declaração for.

Estruturas de repetição: for

```
for (valor_inicial; condição_testada; valor incremento)
{
    comandos;
}
```

- **valor_inicial** refere-se à atribuição de um valor inicial para a variável de controle, por exemplo: “controle = 0;”
- **condição_testada** é uma expressão qualquer contendo a variável de controle, que será testada continuamente. Enquanto a condição for verdadeira, os comandos dentro do laço for serão executados. Quando a condição falhar, o laço termina e o programa continua seu fluxo normal. Por exemplo, “controle < 30;” é uma expressão válida, que testa se a variável controle é menor do que 30.
- **valor_incremento** é uma expressão que incrementará a variável de controle sempre que a condição testada anteriormente for verdadeira. Por exemplo, “controle = controle + 1” ou mais simplesmente “controle++”
- Os **comandos** entre as chaves serão executados sempre que a condição testada for verdadeira, e se repetirão até que a condição se torne falsa e o laço termine.

Estruturas de repetição: for

- **Exemplo**
  - Faça um algoritmo que escreva uma frase 10 vezes.

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int x;
7       for (x=1;x<=10;x++)
8       {
9           cout<<"Esta frase vai se repetir por "<<x<<" vezes\n";
10      }
11      return 0;
12  }
```

  - A variável de controle **x** começa com valor 1.
  - O laço **for** testa o valor da variável continuamente, sempre determinando se ele é menor do que 10. Sempre que o valor da variável for menor que 10 (condição verdadeira), a variável de controle **x** é incrementada de 1 e os comandos entre as chaves são executados.
  - Quando a variável de controle **x** for maior do que 10 (condição falsa), o laço termina e o programa volta a ser executado normalmente.

Exemplos

- **Exemplo:**
  - Calcular a media da altura de 15 pessoas
  - Mostrar o valor da média

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      float alt,media,soma;
7      int x;
8      for (x=1;x<=15;x++)
9      {
10         cout<<"Digite a altura da pessoa "<<x<<":\n";
11         cin>>alt;
12         soma=soma+alt;
13     }
14     media=soma/15;
15     cout<<"A media é "<<media;
16     return 0;
17 }
```

Exemplos

- **Exemplo:** Usando a função **RAND()** para gerar números inteiros aleatórios

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <locale>
using namespace std;

int main()
{
    setlocale(LC_ALL,"portuguese");
    srand(time(NULL));
    int nota=0,aprovado=0,recuperacao=0,reprovado=0;
    for (int i=1;i<=40;i++)
    {
        nota = rand()%10;
        cout<<i<<"° Nota:"<<nota<<endl;
        if(nota >= 6)
        {
            aprovado++;
        }
        if ((nota >= 3) && (nota < 6))
        {
            recuperacao++;
        }
        if (nota < 3)
        {
            reprovado++;
        }
    }
    cout<<"\nNúmero de Alunos Aprovado(s): "<<aprovado<<" aluno(s)\n";
    cout<<"Número de Alunos de Recuperação: "<<recuperacao<<" aluno(s)\n";
    cout<<"Número de Alunos Reprovado(s): "<<reprovado<<" aluno(s)";
    return 0;
}
```

Exercícios

- **Exercício 61**
  - Escreva um algoritmo para ler 10 números e ao final da leitura escrever a soma total dos 10 números lidos.
- **Exercício 62**
  - Ler 10 valores, calcular e escrever a média aritmética desses valores lidos.
- **Exercício 63**
  - Ler 10 valores e escrever quantos desses valores lidos são negativos.
- **Exercício 64**
  - Escreva um algoritmo que calcule e mostre a tabuada de um número N.
- **Exercício 65**
  - Escreva um algoritmo que leia 500 valores inteiros, encontre o maior e o menor valor e calcule a média dos números lidos.
- **Exercício 66**
  - Faça um algoritmo para somar os números pares de 0 até 1000 e ao final imprimir o resultado.

Exercícios

- **Exercício 67**
  - Elabore em algoritmos que analise e mostre os 10 primeiros números inteiros maiores que 100.
- **Exercício 68**
  - Faça um programa que leia 100 valores quaisquer e no final, escreva o maior e o menor valor lido.
- **Exercício 69**
  - Faça um algoritmo que solicite um valor qualquer (quantidade).
  - A seguir leia esta quantidade de números.
  - Depois de ler todos os números o algoritmo deve apresentar na tela:
  1. O maior dos números lidos
  2. A média dos números lidos.
- **Exercício 70**
  - Faça um algoritmo para ler o código e o preço de 15 produtos, calcular e escrever:
  1. O maior preço lido
  2. A média aritmética dos preços dos produtos

Exercícios

- **Exercício 71**
  - Escrever um algoritmo que escreva os números múltiplos de 11 entre 0 e 1500.
  - Mostrar o resultado.
- **Exercício 72**
  - Escreva um algoritmo que calcule e mostre a soma de todos os números pares de 0 a 1000.
- **Exercício 73**
  - Dado a nota final de 40 alunos, faça um algoritmo que:

  a) Verifique a quantidade de alunos aprovados. Média > = 7.0.

  b) Verifique a quantidade de alunos em recuperação. 5.0 >= Média <7.0.

  c) Verifique a quantidade de alunos reprovados. Média <5.0.
- **Exercício 74**
  - Uma loja tem 150 clientes cadastrados e deseja mandar uma correspondência a cada um deles anunciando um bônus especial.
  - Escreva um algoritmo que leia o valor das suas compras no ano passado e calcule um bônus de 10% se o valor das compras for menor que 500.000 e de 15 %, caso contrário.

Exercícios

- **Exercício 75**
  - Escreva um algoritmo que gere o números de 1000 a 1999 e escreva aqueles que dividido por 11 dão resto igual a 5.
- **Exercício 76**
  - Escrever um algoritmo que calcule e mostre a média aritmética dos números lidos entre 13 e 73.
- **Exercício 77**
  - Escrever um algoritmo que gera e escreve os números ímpares entre 100 e 200.
- **Exercício 78**
  - Escreva um algoritmo que leia 50 valores e encontre o maior e o menor deles. Mostre o resultado.
- **Exercício 79**
  - Escrever um algoritmo que leia um conjunto de 50 informações contendo, cada uma delas, a altura e o sexo de uma pessoa (código=1, masculino código=2, feminino), calcule e mostre o seguinte:

  a) a maior e a menor altura da turma

  b) a média da altura das mulheres

Estruturas de repetição: while

- **A declaração WHILE**
  - Uma outra forma de laço é a declaração **while**. Seu funcionamento é muito parecido com a declaração **for** que estudamos anteriormente.
  - Ao encontrar um laço **while**, o programa testa a condição especificada. Se a condição for verdadeira, efetuará os comandos contidos no laço. Quando a condição se torna falsa, o laço termina e o programa passa para o próximo comando.
  - A sintaxe da declaração while é a seguinte:

```
while (condição)
{
    comandos;
}
```

  - A declaração while é diferente é diferente da declaração for em alguns aspectos. Em primeiro lugar, **while** não utiliza variáveis de controle automaticamente.
  - O único argumento entre os parênteses é a condição a ser testada: caso a condição nunca mude e seja sempre verdadeira, estaremos criando um laço infinito. Assim, cabe ao programador inserir uma variável de controle no laço while para evitar que isso ocorra.

Estruturas de repetição: while

- **Exemplo:**
  - Faça um algoritmo que peça na tela vários números positivos infinitamente, até que se coloque um numero negativo, a partir daí mostra na tela a soma dos números positivos e sua media.

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

int main()
{
    float num,media=0,soma=0;
    int x=1,cont=0;
    while (x==1)
    {
        cout<<"Digite um numero:\n";
        cin>>num;
        if (num>=0)
        {
            cont++;
            soma=soma+num;
        }
        else
        {
            x=0;
        }
    }
    media=soma/cont;
    cout<<"A media é "<<media;
    return 0;
}
```

Exercícios

- **Exercício 80**
  - Faça um algoritmo que escreva os números ímpares entre N1 e N2.
- **Exercício 81**
  - Escreva um algoritmo que solicite um número indeterminado de valores positivos e escreva aqueles que dividido por 11 dão resto igual a 5.
  - Encerre o programa digitando **zero** para o número.
- **Exercício 82**
  - Ler a idade e o peso de N pessoas e determinar a média dos pesos das pessoas com mais de 30 anos
  - Encerre o programa digitando **zero** para o peso.
- **Exercício 83**
  - Escreva um algoritmo para ler 2 valores
  - Se o segundo valor informado for **ZERO**, deve ser lido um novo valor.
  - Acrescentar uma mensagem de 'VALOR INVÁLIDO' caso o segundo valor informado seja zero.
  - Mostrar o resultado da divisão do primeiro valor pelo segundo.

Exercícios

- **Exercício 84**
  - Escreva um algoritmo que calcule o volume de uma esfera.
  - Se o valor do raio informado for **ZERO**, deve ser lido um novo valor.
  - Acrescente uma mensagem 'NOVO CÁLCULO (S/N)? ' ao final do cálculo. Se for respondido 'S' deve retornar e executar um novo cálculo, caso contrário deverá encerrar o programa
- **Exercício 85**
  - O país **A** tem 90 milhões de habitantes e uma taxa de natalidade de 3% ao ano
  - O país **B** tem 200 milhões de habitantes e uma taxa de natalidade de 2% ao ano.
  - Calcule e mostre o tempo necessário para que o país **A** ultrapasse a população do país B.
- **Exercício 86**
  - Faça um programa que peça ao usuário um número entre 1 e 20.
  - Se a pessoa digitar um número diferente, mostrar a mensagem "entrada inválida" e solicitar o número novamente.
  - Se digitar correto mostrar o número digitado.

Exercícios

- **Exercício 87**
  - Escreva um algoritmo que faça a soma de 2 números.
  - Acrescente uma mensagem 'NOVO CÁLCULO (S/N)? ' ao final do cálculo. Se for respondido 'S' deve retornar e executar um novo cálculo, caso contrário deverá encerrar o programa
- **Exercício 88**
  - Escreva um algoritmo para ler 2 valores e se o segundo valor informado for **ZERO**, deve ser lido um novo valor. (o segundo valor não pode ser zero)
  - Acrescentar uma mensagem de 'VALOR INVÁLIDO' caso o segundo valor informado seja zero.
  - Imprimir o resultado da divisão do primeiro valor pelo segundo.
- **Exercício 89**
  - Uma loja tem 150 clientes cadastrados e deseja mandar uma correspondência a cada um deles anunciando um bônus especial.
  - Escreva um algoritmo que leia o nome do cliente e o valor das suas compras no ano passado e calcule um bônus de 10% se o valor das compras for menor que 500.000 e de 15 %, caso contrário.

Estruturas de repetição: do while

- **A declaração DO WHILE**
  - A declaração **do while** é muitíssimo parecida com a declaração while, com uma única diferença fundamental: o teste condicional é feito após a execução dos comandos pertencentes ao laço.
  - Veja a sintaxe do laço do while:

```
1  do
2      {
3          comandos;
4      }
5      while (condição);
```

  - Note a inversão da ordem: primeiro temos a declaração **do** seguida do corpo do laço, contendo os comandos à serem executados entre as chaves.
  - Após o símbolo fecha-chaves, temos a declaração while e a expressão que será testada. Os comandos entre as chaves serão executados até que a condição torne-se falsa.
  - Porém, lembre-se que a condição só será testada após a execução dos comandos dentro do laço, ao contrário dos laços que vimos anteriormente que antes testavam uma condição para depois executar qualquer comando.

Estruturas de repetição: do while

- O exemplo abaixo ilustra uma utilização simples do laço do while.
- Neste caso a frase é escrita pelo menos uma vez.
- O sistema testa a condição após escrever a frase.

◆ **Exemplo:**

```
1  #include <iostream>
2  #include <locale>
3  using namespace std;
4
5  int main()
6  {
7      setlocale(LC_ALL,"portuguese");
8
9      int cont=1;
10     do
11     {
12         cout<<"Esta frase foi escrita "<<cont<<" vezes\n";
13         cont++;
14     }
15     while (cont<=20);
16     return 0;
17 }
```

Estruturas de repetição: do while

- No exemplo abaixo o sistema pede o código de acesso até o usuário acertar ou até exceder o número máximo de tentativas.

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

int main()
{
    setlocale(LC_ALL,"portuguese");

    int cod=1234,tentativa=0;
    do
    {
        system ("clear");
        cout<<"Digite o código de acesso:\n";
        cin>>cod;
        if (cod==1234)
        {
            system ("clear");
            cout<<"Acesso permitido";
        }
        else
        {
            tentativa++;
        }
    }
    while (tentativa<=3);
    system ("clear");
    cout<<"Excedeu o número máximo de tentativas permitido.";
    return 0;
}
```

Exercícios

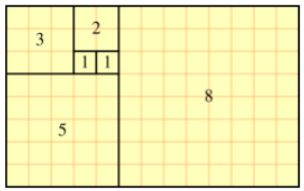
- **Exercício 90**
  - Faça um programa que receba dois números X e Y.
  - Sendo obrigatoriamente X maior que Y, calcule e mostre:
  1. A soma dos números pares desse intervalo, incluindo os números digitados;
  2. A multiplicação dos números ímpares desse intervalo, incluindo os digitados
- **Exercício 91**
  - Leia o valor do salário mínimo atual e o seu nome. Se você poupar o valor do salário mínimo todo mês, verifique quantos anos serão necessários para que você fique milionário. Mostre esta informação.
- **Exercício 92**
  - A prefeitura de uma cidade deseja fazer uma pesquisa entre seus habitantes.
  - Faça um algoritmo para coletar dados sobre o salário e número de filhos de cada habitante e após as leituras, escrever:
  1. Média de salário da população
  2. Média do número de filhos
  3. Menor salário dos habitantes
  4. **Obs**: O final da leituras dos dados se dará com a entrada de um “salário negativo”.

Exercícios

- **Exercício 93**
  - A prefeitura de uma cidade deseja fazer uma pesquisa entre seus habitantes. Faça um algoritmo para coletar dados sobre o salário e número de filhos de cada habitante e após as leituras, escrever:

1. Média de salário da população
2. Média do número de filhos
3. Maior salário dos habitantes
4. **Obs**: O final da leituras dos dados se dará com a entrada de um “salário negativo”.

- **Exercício 94**
  - Sequencia de Fibonacci
  - Considerando os dois primeiros termos da sequência sendo 0 e 1, depois encontre os outros termos somando os seus dois números antecessores. Assim, temos:
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181...
  - Escreva um algoritmo que solicite um número “n” (que deve ser maior que zero) e escreva a sequencia de Fibonacci até este número.
  - Exemplo: se n=100 a sequencia de Fibonacci será:
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89



Aula 9 – Vetores, Matrizes

Professor Ricardo Demattê

IFSP – Unidade Salto

Laços aninhados

Laços aninhados

- Qualquer um dos laços estudados acima pode ser aninhados, ou seja, colocados um dentro do outro.
- Esta técnica pode ser muito útil para trabalhar com **matrizes multidimensionais**, programas que trabalhem com menus e várias outras situações.
- Entretanto, é preciso atenção para não confundir quais blocos fazem parte de qual laço ou declaração.
- É fundamental a **utilização de chaves** para separar corretamente os blocos e laços, e também a **utilização de tabulação** na escrita do código para melhor visualização de cada uma das partes do programa.
- Também é interessante a utilização de **comentários no código**, para melhor identificação.

Vetores

Matrizes de uma dimensão: vetores

- Matrizes são variáveis que contém vários valores de um **mesmo tipo**.
- Por exemplo, podemos criar a matriz notas para armazenar as notas obtidas por 100 alunos em um exame, ou então utilizar uma matriz chamada gastos_mensais para anotar nossos gastos mensais ao longo do ano.
- Uma matriz armazena vários valores de um mesmo tipo: podemos criar matrizes para armazenar qualquer um dos tipos básicos de variáveis, como int, float e char.
- Cada valor é armazenado separadamente em um elemento da matriz, e pode ser acessado e modificado a qualquer momento.
- Para criar uma matriz, precisamos declarar três atributos dela:

1. O **tipo de valor** que vai ser armazenado na matriz
2. O **nome da matriz**, para que possamos acessá-la
3. O **número de elementos** da matriz

Vetores

- A declaração de um matriz é muito parecida com a declaração de uma variável, bastando adicionar o número de elementos que desejamos que ela tenha. A sintaxe é a seguinte:

<tipo> <nome> [<numero de elementos>];

- Por exemplo, caso quiséssemos criar uma matriz chamada catálogo para armazenar 156 inteiros, a declaração seria assim:

int catalogo [156];

- Podemos utilizar qualquer tipo de variáveis já estudadas anteriormente para criar uma matriz, como float, int, char.
- Uma vez criada uma matriz de um determinado tipo, ela só pode receber valores deste tipo.

Vetores

- Assim como uma variável normal, podemos atribuir valores para uma matriz no momento de sua declaração.
- Isto é feito utilizando o operador de atribuição "=" seguido dos valores contidos entre chaves e separados por vírgulas. Por exemplo, considere a matriz de inteiros "teste" abaixo:

int teste[5] = { 1, 2, 3, 4 , 5};

- Também podemos atribuir apenas parte dos valores de uma matriz, por exemplo, podemos criar uma matriz que comporte 50 valores do tipo float e atribuir apenas 5 valores à ela.
- A linguagem C++ faz com que toda matriz parcialmente inicializada tenha seus valores restantes automaticamente transformados em zero.
- Assim, caso precisemos de uma matriz que só contenha zeros, podemos atribuir o primeiro elemento da matriz como zero e deixar que o compilador transforme os elementos restantes em zero, como vemos abaixo:

int zeros[75] = {0};

Vetores

Acessando Valores de uma Matriz

- Após criar uma matriz, podemos acessar qualquer valor dentro dela. Cada valor, ou elemento de uma matriz, possui um número próprio.

- Toda matriz começa no **elemento 0**.
- Precisamos ter isso em mente quando acessamos valores dentro de uma matriz, pois o primeiro elemento será o elemento "**0**", o segundo elemento será o elemento "**1**".

- Cada elemento de uma matriz é tratado como uma variável separada.
- Assim, podemos atribuir valor para um elemento, exibi-lo na tela, utilizá-lo em operações matemáticas e em laços condicionais.

- O programa a seguir ilustra estas várias ações:

Vetores

- **Exemplo:**
  - Exemplo de acesso aos dados de um vetor
  - Criar um vetor de 5 posições
  - Atribuir valor (1,2,3,4 e 5) para cada posição do vetor

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

main()
{
int vetor[5] = {1,2,3,4,5};
cout<<"O primeiro valor do vetor "<<char(130)<<ends<<vetor[0]<<endl;
cout<<"O ultimo valor do vetor "<<char(130)<<ends<<vetor[4]<<endl;
cout<<"Somando o segundo e o quarto elementos do vetor temos: "<< vetor[1] + vetor[3]<<endl;

vetor[2] = 27;

cout<<"Mudamos o valor do terceiro elemento do vetor para: "<<vetor[2]<<endl;

system("PAUSE >> null");
return 0;
}
```

Vetores

Utilizando Laços para Percorrer Matrizes

- Uma das utilizações mais úteis dos laços condicionais é o acesso à vários (ou todos) elementos de uma matriz rapidamente.

- Podemos utilizar qualquer um dos laços que estudamos, mas sem dúvida o laço **FOR** é o mais prático para trabalhar-se com matrizes.

- Utilizamos a **variável de controle** do laço para acessar cada um dos elementos desejados (lembre-se que a matriz sempre começa no elemento 0),
- O programa a seguir percorre todos os elementos de uma matriz, primeiro preenchendo a matriz com os dados entrados pelo usuário, depois exibindo estes dados na tela.
- Para isso utilizamos 2 estruturas de **FOR**, uma para preencher os dados da matriz e outra para exibir estes dados da matriz

Vetores

- **Exemplo:**
  - Inserir e mostrar os dados de um vetor

```
#include <iostream>
using namespace std;
main()
{
int sequencia[4];
//um FOR para inserir dados no vetor
for (int i =  0; i < 4; i++)
{
cout << "Entre com o elemento numero "<<(i+1)<<" da sequencia: ";
cin >> sequencia[i];
cout << endl;
}

//um FOR para mostrar dados no vetor
cout << "A sequencia entrada pelo usuario foi: ";
for (int i = 0; i < 4; i++)
{
cout << sequencia[i]<<" ";
}

system("PAUSE > null");
return 0;
}
```

Vetores

- **Exercício 78**
  - Leia os vetores A (20 elementos) e B (20 elementos).
  - Construa o vetor C e D.
  - Cada elemento de C é a subtração do elemento correspondente de A–B.
  - Cada elemento de D é a multiplicação do elemento correspondente de AxB.
  - Imprima os vetores A,B,C e D.
- **Exercício 79**
  - Uma prova de química foi feita por um grupo de 20 alunos.
  - Faça um algoritmo para ler a nota dos 20 alunos.
  - Ao final, emita um relatório com as 20 notas dos alunos e a média global deste grupo de alunos.
- **Exercício 80**
  - Escrever um algoritmo que leia o nome, o sexo, a idade e altura de 5 pessoas.
  - Exiba o total de homens e de mulheres.
  - Exiba uma lista de nomes das pessoas aptas a prestar o serviço militar.
  - Obs: Somente são aptos os homens, maiores de 18 anos, com altura maior ou igual a 1,70.

Matrizes

Matrizes multidimensionais

- Além das matrizes simples de uma única dimensão, C++ permite a criação de matrizes de múltiplas dimensões.
- As matrizes bidimensionais são sem dúvida as mais utilizadas e as mais úteis, pois comportam-se como tabelas com linhas e colunas.
- Ao declarar uma matriz multidimensional, adicionamos um conjunto de colchetes para cada dimensão extra.
- Entre os colchetes de cada dimensão, colocamos o número de elementos que aquela dimensão terá (ou uma variável que represente o número de elementos).
- Assim:

```
int tabela [10] [5]; //matriz bidimensional
int horas [12] [30] [24]; //matriz de três dimensões
int minutos [12] [30] [24] [60]; //matriz de quatro dimensões
```

Matrizes

Matrizes multidimensionais

- Normalmente trabalhamos no máximo com matrizes bidimensionais, mas podem surgir ocasiões onde matrizes de mais de duas dimensões sejam necessárias.
- Matrizes multidimensionais funcionam como **matrizes dentro de matrizes**.
- Por exemplo, uma matriz bidimensional pode ser vista como uma matriz de uma dimensão cujos elementos são outra matrizes.
- Esta analogia é útil para entender como é feita a inicialização dos valores de matrizes multidimensionais: inicializamos a matriz separando seus elementos por vírgulas, onde cada elemento é uma matriz individual e é inicializada da mesma forma que a matriz “principal”.
- Por exemplo, seja a matriz bidimensional tabela:

```
int tabela [2] [3] = { { 1, 2, 3} , { 4, 5, 6}};
```

Matrizes

Matrizes multidimensionais

- Normalmente trabalhamos no máximo com matrizes bidimensionais, mas podem surgir ocasiões onde matrizes de mais de duas dimensões sejam necessárias.
- Matrizes multidimensionais funcionam como **matrizes dentro de matrizes**.
- Por exemplo, uma matriz bidimensional pode ser vista como uma matriz de uma dimensão cujos elementos são outra matrizes.
- Esta analogia é útil para entender como é feita a inicialização dos valores de matrizes multidimensionais: inicializamos a matriz separando seus elementos por vírgulas, onde cada elemento é uma matriz individual e é inicializada da mesma forma que a matriz “principal”.
- Por exemplo, seja a matriz bidimensional tabela:

```
int tabela [2] [3] = { { 1, 2, 3} , { 4, 5, 6}};
```

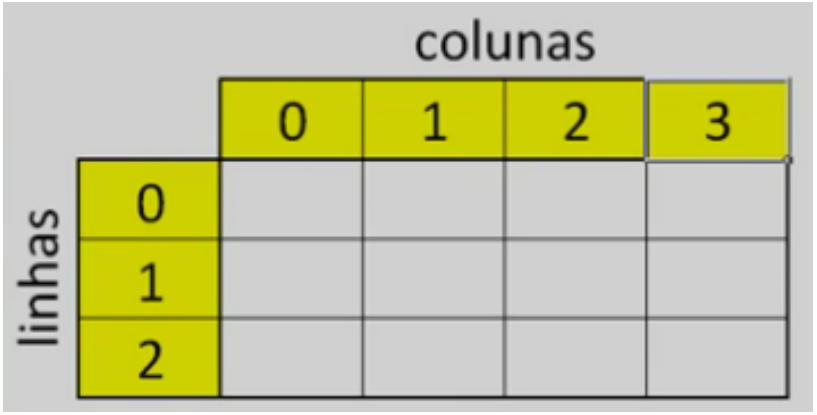
Matrizes

- A declaração de um matriz é muito parecida com a declaração de uma variável, bastando adicionar o número de elementos que desejamos que ela tenha. A sintaxe é a seguinte:

<tipo> <nome> [<numero de elementos>] [<numero de elementos>];

- Por exemplo, caso quiséssemos criar uma matriz chamada notas para armazenar as notas de 4 bimestres de 3 alunos, a declaração seria assim:

float notas [3] [4];



Matrizes

- **Exemplo:**
  - Escrever um algoritmo que **somente** leia as notas de 4 bimestres de 3 alunos.

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

main()
{
    float notas[3][4];
    int l,c;
    for (l=0;l<3;l++)
    {
      for (c=0;c<4;c++)
      {
        cout <<"Digite a "<<c+1<<" nota do aluno "<<l+1<<endl;
        cin >> notas[l][c];
       }
    }
    cout << notas[4][3] << endl;
    system ("pause>>null");
    return 0;
}
```

Matrizes

- **Exemplo:**
  - Escrever um algoritmo que leia as notas de 4 bimestres de 3 alunos.
  - Mostre as notas que cada aluno tirou em cada bimestre

```
#include <iostream>
using namespace std;
main()
{
    float notas[3][4];
    int l,c;
    // cadastrar notas
    for (l=0;l<3;l++)
    {
      for (c=0;c<4;c++)
      {
        cout <<"Digite a "<<c+1<<" nota do aluno "<<l+1<<endl;
        cin >> notas[l][c];
       }
    }
    system ("cls");
    // mostrar notas
    for (l=0;l<3;l++)
    {
      for (c=0;c<4;c++)
      {
        cout <<"A nota do "<<c+1<<" bimestre do aluno "<<l+1<<" = "<<notas[l][c]<<endl;
      }
    }
    system ("pause>>null");
    return 0;
}
```

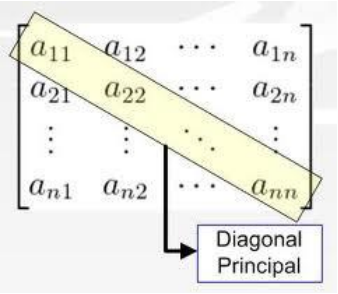
Matrizes

- **Exemplo:**
  - Escrever um algoritmo que cadastre valores em uma matriz 3x3
  - Mostre todos os elementos da matriz

```
#include <iostream>
using namespace std;
main()
{
     int matriz[3][3];
     int l,c;
     // cadastrar matriz
     for (l=0;l<3;l++)
       for (c=0;c<3;c++)
       {
         cout <<"Digite o elemento da posicao "<<l+1<<c+1<<endl;
         cin >> matriz[l][c];
       }
     system ("cls");
     // mostrar os elementos da matriz
     for (l=0;l<3;l++)
       for (c=0;c<3;c++)
       {
         cout <<"O valor do elemento da posicao "<<l+1<<c+1<<" = "<<matriz[l][c]<<endl;
       }
     system ("pause>>null");
     return 0;
}
```

Matrizes

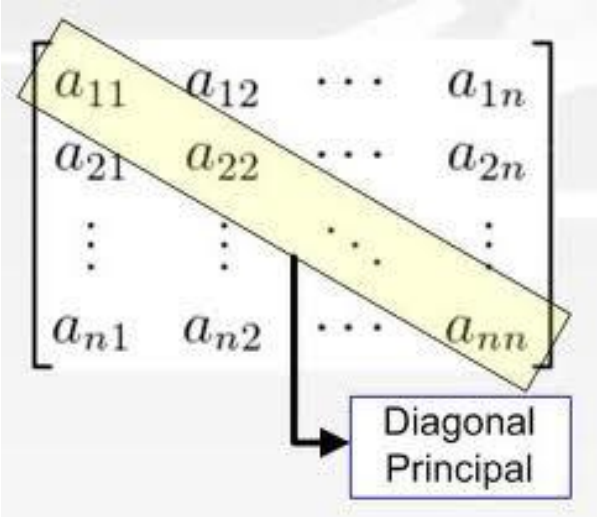
- **Exemplo:**
  - Escrever um algoritmo que cadastre valores em uma matriz 3x3
  - Apresente a soma dos elementos da diagonal principal




```
#include <iostream>
using namespace std;
main()
{
     int diag[3][3];
     int l,c;
     int diag_pri=0;
     // cadastrar matriz
     for (l=0;l<3;l++)
       for (c=0;c<3;c++)
       {
         cout <<"Digite o elemento da posicao "<<l+1<<c+1<<endl;
         cin >> diag[l][c];
         // calcular elementos da diagonal principal
         if (l==c)
         diag_pri += diag[l][c];
       }
     system ("cls");
     // mostrar a soma dos elementos da diagonal
     cout <<"A soma dos elementos da diagonal principal = "<<diag_pri<<endl;
     system ("pause>>null");
     return 0;
}
```

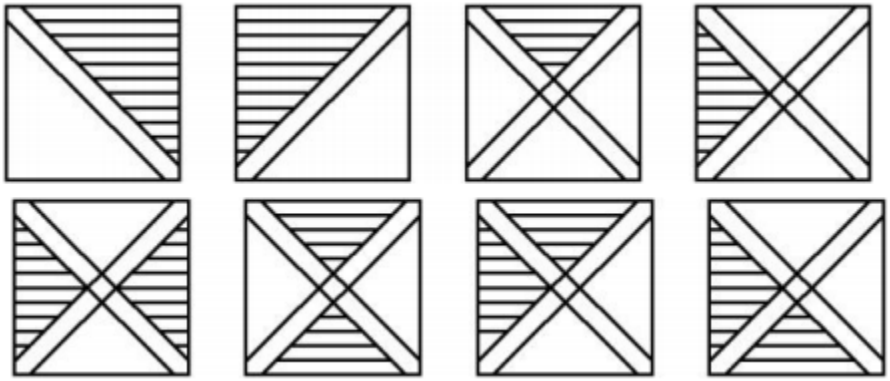
Matrizes

- **Exemplo:**
  - Escrever um algoritmo que cadastre valores em uma matriz 3x3
  - Apresente a soma dos elementos da diagonal invertida



Matrizes

- **Exercício 81**
- Escrever um algoritmo que lê uma matriz M(6,6) e calcule as somas das partes hachuradas.
  - Escrever a matriz M e as somas calculadas.



Matrizes

- **Exercício 82**
  - Escrever um algoritmo para armazenar valores inteiros em uma matriz (5,6).
  - A seguir, calcular a média dos valores pares contidos na matriz e escrever seu conteúdo.

- **Exercício 83**
  - Escrever um algoritmo para ler uma matriz (7,4) contendo valores inteiros (supor que os valores são distintos).
  - Após, encontrar o menor valor contido na matriz e sua posição.

- **Exercício 84**
  - Escrever um algoritmo que lê duas matrizes N1(4,6) e N2(4,6) e crie:
  - Uma matriz M1 que seja a soma de N1 e N2
  - Uma matriz M2 que seja a diferença de N1 com N2
  - Escrever as matrizes lidas e calculadas.

Matrizes

- **Exercício 85**

  - Escrever um algoritmo que leia as notas de 4 bimestres de 20 alunos.
  - Solicite o nome do aluno
  - Calcule a média de cada aluno
  - Se a média do aluno for maior ou igual a 6 o aluno esta aprovado
  - Se média menor que 6, aluno está reprovado
  - Ao final escreva:

  1. O nome do aluno
  2. A nota que este aluno tirou nos 4 bimestres
  3. A sua média final
  4. E se ele está aprovado ou reprovado

Aula 10 – Funções

Professor Ricardo Demattê

IFSP – Unidade Salto

INSTITUTO FEDERAL
São Paulo
Câmpus Salto

Funções

Funções

- Funções são os blocos de construção da linguagem C++, com os quais podemos construir programas melhores e mais facilmente compreensíveis.
- Quando os programas tornam-se **maiores** e mais **complexos**, pode-se melhorar a clareza e compreensão do trabalho dividindo-o em partes menores, que chamamos de funções.
- Todo programa possui ao menos uma função: a função **main**, a qual podemos chamar de “corpo” do programa, onde estão localizados todos os comandos e chamadas de outras funções que são executadas pelo programa.
- Além da função **main**, podemos utilizar e também criar várias outras funções dentro do mesmo programa.

Funções

Funções

- Por exemplo, imagine um programa que organize o funcionamento de uma loja.
- Poderia haver uma função para **organizar o estoque**, outra para relacionar os **preços dos produtos**, outra para **acessar um banco de dados** com os **cadastros dos clientes**, entre outras.
- Se fossem colocados todos os comandos do programa dentro da função **main**, o programa ficaria muito grande e provavelmente incompreensível.
- Dividindo o programa em funções separadas, deixando para a função **main** somente a tarefa de organizar as chamadas das demais funções, podemos trabalhar mais facilmente com o programa, modificando e corrigindo funções individualmente.
- À medida que o tamanho e a complexidade do programa aumentam, aumenta também a possibilidade de erros, já se o mesmo for dividido em blocos menores e organizados, fica mais fácil encontrar e evitar erros.

Funções

Funções

- Basicamente, uma função funciona da seguinte forma:

1. é feita uma chamada para esta função durante a execução do programa;
2. o programa é interrompido temporariamente, e “pula” para esta função executando seus comandos;
3. quando a função termina, ou seja, quando seus comandos acabam (ou quando o comando return é encontrado), o programa volta ao ponto onde foi interrompido para continuar sua execução normal.

- Uma função pode receber dados do programa para executar seus comandos (estes dados são chamados de parâmetros ou argumentos), e pode também retornar dados para o programa (o que chamamos de retorno de função).

Funções

Declarando uma função

- Antes de utilizar uma função em um programa é preciso declará-la, ou seja, especificar para o compilador exatamente o que faz esta função e que tipo de dados ela recebe e retorna.
- Podemos dividir o processo de declarar uma função em duas etapas distintas: o **protótipo da função** e a **definição da função**.

Protótipo de uma função

- O protótipo de uma função tem a mesma função de uma declaração de variáveis: dizer ao compilador quais tipos de variáveis estarão envolvidos na função.
- Isso permite que o compilador saiba trabalhar corretamente com os valores que entram e saem da função, fazendo conversões de tipos sempre que necessário.

Funções

Protótipo de uma função

- A sintaxe da declaração de um protótipo de função é a seguinte:

```
<tipo_da_função> <nome> ( <tipo_de_parâmetro> <nome_da_variável>);
```

- O **tipo da função** denominará qual será o **tipo de valor que esta função retorna**. Uma função pode retornar qualquer tipo de valor, seja inteiro, fracionário ou nenhum valor.
- Quando uma função não retorna nenhum valor para o programa, seu tipo é **void**. Esse tipo de função executará seus comandos normalmente, mas não retornará nenhum valor para o programa quando terminar.
- Já o **tipo de parâmetro** serve para determinar o **tipo de variáveis que a função receberá** como parâmetros. Assim como o tipo da função, qualquer tipo de variável pode ser utilizado como parâmetro.
- Uma função que não recebe parâmetros de entrada deve ter a palavra chave **void** entre parênteses. É possível ter vários parâmetros na mesma função, separando-os com vírgulas.

Funções

Protótipo de uma função

- A sintaxe da declaração de um protótipo de função é a seguinte:

```
<tipo_da_função> <nome> ( <tipo_de_parâmetro> <nome_da_variável>);
```

- O **tipo da função** denominará qual será o **tipo de valor que esta função retorna**.
- O **tipo de parâmetro** serve para determinar o **tipo de variáveis que a função receberá** como parâmetros.
- O **void** não recebe ou retorna valores.
- Alguns exemplos de protótipo de função:

int livro (**unsigned int paginas**);

float divide (**float dividendo, float divisor**);

void imprime (**void**);

float divide (**float, float**);

Funções

Definição de uma função

- A definição de uma função diz ao compilador exatamente o que a função faz. A sintaxe da definição de uma função é a seguinte:

```
<tipo da função> <nome> ( <tipo do parâmetro> <nome do parâmetro>)
{
comandos da função;
}
```

- A primeira linha da definição de uma função é idêntica ao protótipo, com a exceção de que somos obrigados a declarar nomes para as variáveis de parâmetro e que a linha não termina em um ponto-e-vírgula, e sim em uma abre-chaves.
- Dentro das chaves estarão todos os comandos pertencentes a função, inclusive declaração de variáveis locais, chamadas para outras funções e chamadas de retorno.

Funções

Definição de uma função

- Por exemplo:

```
int cubo (int valor)
{
resultado = valor*valor*valor;
return resultado;
}
```

- A função acima calcula o cubo de um valor inteiro.
- Note que podemos declarar variáveis dentro de uma função; entretanto, estas variáveis só poderão ser utilizadas **dentro desta mesma função**.
- O comando return termina a função e retorna um valor para o programa.

Funções

Retorno de uma função

- O comando return é utilizado para terminar a execução de uma função e retornar um valor para o programa.
- Sua sintaxe é a seguinte:

```
return <variável ou expressão>;
```

- O comando return aceita qualquer constante, variável ou expressão geral que o programador precise retornar para o programa principal, desde que este valor seja igual ou convertível para o tipo da função (já estabelecido no protótipo da função).
- É importante notar que o valor retornado não pode ser uma matriz;
- É possível também criar funções que contenham múltiplos comandos return, cada um dos quais retornando um valor para uma condição específica.

Funções

Retorno de uma função

- Por exemplo, considere a função **compara_valores**, mostrada a seguir:

```
int compara_valores(int primeiro, int segundo)
{
if (primeiro == segundo)
return (0);
else if (primeiro > segundo)
return (1);
else if (primeiro < segundo)
return (2);
```

- A função **compara_valores** examina dois valores listados na tabela abaixo:

Resultado	Significado
0	Os valores são iguais.
1	O primeiro valor é maior que o segundo.
2	O segundo valor é maior que o primeiro.

Funções

Variáveis dentro de uma função

- Para usar uma variável dentro de uma função, precisa-se primeiro declarar a variável, exatamente como feito na função principal main.
- Quando se declara variáveis dentro de uma função, os nomes usados para essas variáveis **são exclusivos para a função**.

Variáveis locais

- A Linguagem C++ permite declarar variáveis dentro de suas funções.
- Essas variáveis são chamadas de variáveis locais, pois seus nomes e valores somente têm significado dentro da função que contém a declaração da variável.

Variáveis globais

- Além das variáveis locais, a Linguagem C++ permite que os programas usem variáveis globais, cujos nomes, valores e existência são conhecidos em todo o programa.
- É preciso estar atento com o uso de variáveis globais e sua nomeação. Quando nomes de variáveis globais e locais estiverem em conflito, a linguagem C++ usará sempre a variável **local**.

Funções

- **Variáveis locais**

```
#include <iostream>
using namespace std;
void valores_locais(void);
void valores_locais(void)
{
int a=1, b=2, c=3;
cout<<"A vale "<<a<<".  B vale "<<b<<". C vale "<<c<<".";
}
int main (void)
{
valores_locais();
system("PAUSE > null");
return 0;
}
```

- **Variáveis globais**

```
#include <iostream>
using namespace std;
int a = 1, b = 2, c = 3;    // Variaveis globais
void valores_globais(void)
{
cout<<"A vale "<< a <<". B vale "<< b <<". C vale "<< c <<".\n";
}
int main(void)
{
valores_globais();
cout<<"A vale "<< a <<". B vale "<< b <<". C vale "<< c <<".\n";
system("PAUSE > null");
return 0;
}
```

Funções

Main como função

- Todo programa possui uma função principal que contém todos os comandos e chamadas para outras funções presentes no programa.
- A função **main** funciona como uma função normal: possui um protótipo e uma definição. Geralmente omitimos o protótipo, fazendo apenas a definição da função main da seguinte forma:

```
int main (void) { corpo do programa return 0; }
```

- A função main é do tipo int, e retorna 0. Entretanto, não existe outra função acima de main que a tenha chamado, para que ela possa retornar um valor de resposta. Para que serve este retorno então? Simples: consideramos que a "função chamadora" de main é o próprio sistema operacional.
- Assim, utilizamos o retorno para indicar o funcionamento do programa. Caso o programa termine e retorne o valor **0** para o sistema operacional, sabemos que tudo correu bem e que o programa terminou normalmente.
- Um valor retornado **diferente de 0** indica que o programa não rodou até o final (ou seja, até o ponto "return 0;") e que aconteceu algum erro.

Funções

- **Exemplo:**
  - Escrever um algoritmo para mostrar a mensagem “Sou aluno do IFSP” na tela usando função
  - Sem retorno a função principal (void)

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

void mensagem() // função mensagem
{
cout <<"Sou aluno do IFSP"<<endl;
system("PAUSE > null");
}

main()  //função principal
{
mensagem(); // chamada da função mensagem
return 0;
}
```

Funções

- **Exemplo:**
  - Escrever um algoritmo que contenha uma função “calcular”
  - Esta função retorna o valor 10 (inteiro)
  - Mostrar um valor de retorno desta função.

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

int calcular()
{ return 10; }

int main()
{
cout<<"Resultado: "<<calcular();
system("PAUSE > null");
return 0;
}
```

Funções

- **Exemplo:**
  - Escrever um algoritmo que contenha uma função “somar” 2 números
  - Executar e mostrar o resultado. Sem retorno a função principal (void)

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

void somar() //função somar sem retorno
{
float num1,num2,soma;
cout<<"Digite um numero: "<<endl;
cin>>num1;
cout<<"Digite outro numero: "<<endl;
cin>>num2;
soma=num1+num2;
cout<<"O resultado da soma = "<<soma<<endl;
}

int main()  //função principal
{
somar();
system("PAUSE > null");
return 0;
}
```

Funções

- **Exemplo:**
  - Escrever um algoritmo para verificar se um número é par ou impar. Use uma função para esta verificação. Na função, se for par retorne o valor 1, se for impar retorne 0. Mostre o resultado.

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

int par(int x) // função para verificar se número é par
{
  if (x%2==0)
   {return 1;}
  else
   {return 0;}
}

main()  //função principal
{
int num;
cout<<"Digite um numero:"<<endl;
cin>>num;

if (par(num)==1) // chamada da função PAR
{cout<<"O numero "<<num<<" e' PAR"<<endl;}
else
{cout<<"O numero "<<num<<" e' IMPAR"<<endl;}

system("PAUSE > null");
return 0;
}
```

Funções

- **Exemplo:**
  - Faça a mesma verificação do exercício anterior, utilizando um vetor de 10 posições, analise número a número e mostre o resultado.

```
#include <iostream>
using namespace std;
int par(int n) // função para verificar se número é par
{
  if (n%2==0)
   (return 1;)
  else
   (return 0;)
}
main()  //função principal
{
int x;
int vetor[10];
for (x=0;x<10;x++)  // inserir os números no vetor
{
    cout<<"Digite um numero:"<<endl;
    cin>>vetor[x];
}
system("CLS");
for (x=0;x<10;x++)  // verificar se é par ou impar
{
    if (par(x)==1) //chamada da função
    (cout<<"O numero "<<vetor[x]<<" e' PAR"<<endl;)
    else
    (cout<<"O numero "<<vetor[x]<<" e' IMPAR"<<endl;)
}
system("PAUSE > null");
return 0;
}
```

Funções

- **Exemplo:**
  - Usando funções, desenvolva um algoritmo que implemente uma calculadora que realize operações com 2 números quaisquer.
  - A calculadora deve realizar as seguintes operações, separadamente:

1. Soma
2. Subtração
3. Divisão
4. Multiplicação
5. Potenciação

  - Faça um menu de opções
  - Não pode haver divisão por 0 (zero)
  - Mostre o resultado de cada operação realizada.
  - Após realizada uma operação, pergunte se deseja realizar nova operação ou encerrar o programa.

Funções

- **Resposta do exemplo**
  - **Declaração das funções do sistema**

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

// funções do sistema

void menu();
void adicao();
void subtracao();
void multiplicacao();
void divisao();
void sair();

int main(void)
{
    menu();
    system ("pause>>null");
    return 0;
}
```

Funções

- **Resposta do exemplo**
  - **Menu do sistema**

```
void menu()
{
int opcao;
cout<<"Informe sua opcao:"<<endl;
cout<<"1 - Para adicao:"<<endl;
cout<<"2 - Para subtracao:"<<endl;
cout<<"3 - Para multiplicacao:"<<endl;
cout<<"4 - Para divisao"<<endl;
cout<<"5 - Para sair:"<<endl;
cin>>opcao;
```

Funções

- **Resposta do exemplo**
  - **Menu do sistema usando switch**

```
switch (opcao)
 {
   case 1:
     adicao();
     break;
   case 2:
     subtracao();
     break;
   case 3:
     multiplicacao();
     break;
   case 4:
     divisao();
     break;
   case 5:
     sair();
     break;
   default:
   cout<<"Opcao invalida! Selecione novamente"<<endl;
   system ("pause");
   system ("cls");
   menu();
 }
}
```

Funções

- **Resposta do exemplo**
  - **Função adição**

```
void adicao ()
{
float n1,n2,soma;
cout<<"Voce selecionou a funcao de adicao"<<endl<<endl;
system ("pause");
system ("cls");
cout<<"Digite o primeiro numero"<<endl;
cin>>n1;
cout<<"Digite o segundo numero"<<endl;
cin>>n2;
soma = n1+n2;
cout<<"O valor da soma de "<<n1<<" + "<<n2<<" = "<<soma<<endl<<endl;
system ("pause");
system ("cls");
menu();
}
```

Funções

- **Resposta do exemplo**
  - **Outras funções do sistema**

```
void subtracao ()
{
cout<<"Voce selecionou a funcao de subtracao"<<endl;
menu();
}

void multiplicacao ()
{
cout<<"Voce selecionou a funcao de multiplicacao"<<endl;
menu();
}

void divisao ()
{
cout<<"Voce selecionou a funcao de divisao"<<endl;
menu();
}

void sair ()
{
exit(1);
}
```

Aula 11 - Strings

Professor Ricardo Demattê

IFSP – Unidade Salto

Strings

Caracteres e Strings

- Os caracteres são entendidos como sendo números que geralmente têm oito bits, esses números são traduzidos na tabela ASCII de 128 caracteres, como existem inúmeras regiões no mundo com características linguísticas próprias, a tabela ASCII é estendida por um bloco de caracteres acima dos 128 mais baixos que varia de acordo com as necessidades de cada língua.
- A parte superior da tabela ASCII é conhecida como parte estendida e é referenciada por páginas de códigos para cada propósito linguístico, isso quer dizer que podemos ter os mesmos números significando caracteres diferentes para cada região do mundo.
- No estilo da linguagem C quando queremos representar um conjunto de caracteres colocamos todos eles em uma matriz sequenciada na memória:

Endereço relativo	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A
Dado	U	m	a		f	r	a	s	e	.

- Por exemplo, para declarar um espaço na memória que contenha 20 caracteres fazemos:

```
char dados[20];
```

Strings

Strings em C

- Este é o estilo de strings usado pela linguagem C pura:

```
char dados[20];
```

- Desta forma podemos classificar as strings como vetores de caracteres ou **arrays** de *chars ou caracteres*. Strings são o uso mais comum para os vetores.
- Devemos apenas ficar atentos para o fato de que as strings tem o seu último elemento como um '\0'. A declaração geral para uma string é:

```
char nome_da_string [tamanho];
```

- Devemos lembrar que o tamanho da string deve incluir o '\0' final.
- A biblioteca padrão do C possui diversas funções que manipulam strings. Estas funções são úteis pois não se pode, por exemplo, igualar duas strings:

```
string1=string2;          /* NAO faca isto */
```

- As strings devem ser igualadas elemento a elemento, pois são vetores de caracteres.

Strings

- Quando vamos fazer programas que tratam de string muitas vezes podemos fazer bom proveito do fato de que uma string termina com '\0' (isto é, o número inteiro 0).
- Veja, por exemplo, o programa a seguir que serve para igualar duas strings (isto é, copia os caracteres de uma string para o vetor da outra):

```
#include <iostream>
int main ()
{
    int count;
    char str1[100],str2[100];
    ....            /* Aqui o programa le str1 que sera copiada para str2 */
    for (count=0;str1[count];count++)
            str2[count]=str1[count];
        str2[count]='\0';
....            /* Aqui o programa continua */
}
```

- A condição no loop for acima é baseada no fato de que a string que está sendo copiada termina em '\0'.
- Quando o elemento encontrado em str1[count] é o '\0', o valor retornado para o teste condicional é falso (nulo).
- Desta forma a expressão que vinha sendo verdadeira (não zero) continuamente, torna-se falsa.

Strings

Strings em C++

- As cadeias de caracteres da **linguagem C** podem formatar um novo tipo de dados, porém criar tipos de dados mais sofisticados não é possível nesta linguagem.
- As strings em **C++** são objetos da classe **string**, e a primeira coisa a notar quando criamos strings em C++ é a maneira de criá-las, a classe disponibiliza uma série de construtores e isto torna possível, basicamente, criar string de seis maneiras diferentes:

1. Podemos definir um objeto string vazio, para futuramente usarmos de acordo com a necessidade;
2. Podemos criar um objeto string com uma cópia de outro;
3. Podemos criar um objeto string com uma cópia de uma porção de outra string;
4. Podemos criar um objeto string com uma cópia de uma parte de uma "char string";
5. Podemos criar um objeto string com uma cópia de uma "char string";
6. Podemos criar um objeto string preenchida com uma quantidade definida de um determinado caractere;

Strings

Strings em C++

- Quando manipulamos strings, podemos fazê-lo com operadores, como por exemplo "+", "+=", "<<", etc...
- Isto torna o código um pouco mais intuitivo, vejamos os operadores e suas respectivas operações:

- Operador **=** Atribuir o valor de uma string para outra;
- Operador **[ ]** Acessar caracteres individualmente;
- Operador **+=** Adicionar uma string no final de outra;
- Operador **+** Concatenar strings;
- Operador **<<** Enviar uma string a um output stream;
- Operador **>>** Receber uma string do input stream.

Strings

- **Exemplo:**

```
#include <iostream>
#include <string.h>
using namespace std;
int main(void)
{
string a = "Alice e Beto gostam de ",
b("chocolate."),
c = string("doce de leite."),
d = "pipoca.",
e(c);

cout << a + b << endl;
cout << a + c << endl;
cout << a + d << endl;
cout << a + e << endl;
system ("pause>>null");
return 0;
}
```

Strings

- **Funções de manipulação de strings - <string.h>**
  - **strcpy()**
  - A função strcpy() copia a string origem para a string destino.
  - A função strcpy() pertence a biblioteca **<string.h>**.
  - O programa abaixo demonstra o funcionamento da função strcpy():

```
#include <iostream>
using namespace std;
#include <string.h>
int main ()
{
     char str1[100],str2[100],str3[100];
     cout<<"Entre com uma string: "<<endl;
     gets (str1);
     strcpy (str2,str1); /* Copia str1 em str2 */
     strcpy (str3,"Voce digitou a string "); /* Copia "Voce digitou a string" em str3 */
     cout<<str3<<str2;
     system ("pause>>null");
     return(0);
}
```

Strings

- **Funções de manipulação de strings - <string.h>**
  - **strcat()**
  - A função strcat() concatena ou soma o valor das strings.
  - A função strcat() pertence a biblioteca **<string.h>**.
  - O programa abaixo demonstra o funcionamento da função strcat():

```
#include <string.h>
#include <iostream>
using namespace std;
int main ()
{
     char str1[100],str2[100];
     cout<<"Entre com uma string: "<<endl;
     gets (str1);
     strcpy (str2,"Voce digitou a string "); // copia a frase para str2
     strcat (str2,str1); /* str2 armazenara' Voce digitou a string + o conteudo de str1 */
     cout<<str2<<endl;
     system ("pause>>null");
     return(0);
}
```

Strings

- **Funções de manipulação de strings - <string.h>**
  - **strlen()**
  - A função strlen() retorna o comprimento da string fornecida
  - A função strlen() pertence a biblioteca **<string.h>**.
  - O programa abaixo demonstra o funcionamento da função strlen():

```
#include <string.h>
#include <iostream>
using namespace std;
int main ()
{
     int size;
     char str[100];
     cout<<"Entre com uma string: "<<endl;
     gets (str);
     size=strlen (str);
     cout<<"A string que voce digitou tem tamanho "<<size<<endl;
     system ("pause>>null");
     return(0);
 }
```

Strings

- **Funções de manipulação de strings - <string.h>**
  - **strcmp()**
  - A função strcmp() compara a string 1 com a string 2.
  - Se as duas forem idênticas a função retorna zero. Se elas forem diferentes a função retorna não-zero. A função strcmp() pertence a biblioteca **<string.h>**.
  - O programa abaixo demonstra o funcionamento da função strcmp():

```
#include <string.h>
#include <iostream>
using namespace std;
int main ()
{
    char str1[100],str2[100];
    cout<<"Entre com uma string: "<<endl;
    gets (str1);
    cout<<"Entre com outra string: "<<endl;
    gets (str2);
    if (strcmp(str1,str2))
        cout<<"As duas strings sao diferentes."<<endl;
    else
        cout<<"As duas strings sao iguais."<<endl;
    system ("pause>>null");
    return(0);
 }
```

Rand

- **Função rand, random e srand**
  - No C/C++ é usado um gerador de números aleatórios para facilitar a operação e teste de sistemas.
  - A função rand e srand serve para gerar números aleatórios
- **Exemplo:**

```
#include<iostream>
using namespace std;
//* as funções rand(), random() e srand() estão na biblioteca iostream ou stdlib.h *
main()
{
int i;
/* inicializar o gerador de números aleatórios */
srand(time(NULL));
for (i=0; i<5; i++)
{
/* para gerar números aleatórios de 0 a 50 */
cout <<rand()%50<<endl;
}
system("pause>>null");
return 0;
}
```

Exercícios Complementares

Professor Ricardo Demattê

IFSP – Unidade Salto

INSTITUTO FEDERAL
São Paulo
Câmpus Salto

Exercícios

- **Exercício 86 - Cálculo da área do retângulo.**
  1. Utilize um comando de repetição que permita realizar este cálculo quantas vezes você desejar, em uma estrutura de repetição, até que você digite um número negativo para parar de executar este cálculo.
  2. Utilize um comando de repetição que permita realizar este cálculo por 100 vezes.
  3. Utilize um comando de repetição que permita realizar este cálculo por “n” vezes, até que você digite 0 (zero). Conte quantas áreas de retângulos são maiores que 5000 e mostre este resultado.
- **Exercício 87 – Verifique se o número é par ou impar.**
  1. Utilize um comando de repetição que permita realizar este cálculo por 10 vezes. Conte quantos números pares e impares foram digitados e mostre este resultado.
  2. Utilize um comando de repetição que permita realizar este cálculo por n vezes, até que você digite um número negativo. Conte quantos números pares e impares foram digitados e mostre este resultado.
  3. Utilize um comando de repetição que permita realizar este cálculo por “n” vezes, até que você digite um número negativo. Conte quantos números pares e impares foram digitados e mostre este resultado. Pergunte se deseja executar novamente este programa.

Exercícios

- **Exercício 88**
  - Escreva um algoritmo que leia dois vetores de 10 posições e faça a multiplicação dos elementos de mesmo índice, colocando o resultado em um terceiro vetor. Mostre o vetor resultante.
- **Exercício 89**
  - Escreva um algoritmo que leia e mostre um vetor de 20 elementos inteiros.
  - A seguir, conte quantos valores pares existem no vetor.
- **Exercício 90**
  - Em um vetor com 7 posições, inserir um texto em cada posição e ao final, mostrar todos os textos na tela, em sequencia.
- **Exercício 91**
  - Escreva um algoritmo que leia um vetor de 7 elementos inteiros. Encontre e mostre o menor elemento e sua posição no vetor.
- **Exercício 92**
  - Escreva um algoritmo que leia um vetor de 500 posições de números inteiros e divida todos os seus elementos pelo maior valor do vetor.
  - Realize os cálculos e mostre o resultado em um outro vetor.

Exercícios

- **Exercício 93**
  - Leia uma matriz 10 x 10 e escreva a localização (linha e a coluna) do maior valor.
- **Exercício 94**
  - Declare uma matriz 5 x 5.
  - Preencha com 1 a diagonal principal e com 0 os demais elementos.
  - Escreva ao final a matriz obtida.
- **Exercício 95**
  - Leia duas matrizes 4 x 4
  - Escreva uma terceira matriz com a soma dos valores de mesmo índice.
  - Escreva uma quarta matriz com a multiplicação dos valores de mesmo índice.
- **Exercício 96**
  - Leia uma matriz 6 x 6, conte e escreva quantos valores maiores que 10 ela possui.
- **Exercício 97**
  - Leia uma matriz 20 x 20. Leia também um valor X.
  - O programa deverá fazer uma busca desse valor na matriz e, ao final escrever a localização (linha e coluna) ou uma mensagem de “não encontrado”.

Exercícios

- **Exercício 98**
  - Escreva um algoritmo que utilize um vetor para cadastra o nome de 40 alunos e utilize uma matriz que cadastre a nota destes em todos os bimestres do ano:
  - 1º, 2º, 3º e 4º bimestre.
  - Mostre o nome do aluno e suas respectivas notas
- **Exercício 99**
- Escreva um algoritmo que utilize uma matriz que cadastre a nota de 20 alunos em todos os bimestres do ano: 1º, 2º, 3º e 4º bimestre.
  - Calcule a média global dos alunos desta sala
- **Exercício 100**
  - Um distribuidor de refrigerantes vende seu produto por todo o país.
  - Em cada trimestre do ano passado ele vendeu uma certa quantidade de garrafas em cada região do Brasil.
  - As 5 regiões do Brasil são: Norte, Nordeste, Centro-Oeste, Sudeste e Sul.
  - Faça um algoritmo para ler as quantidades vendidas por região e escrever a quantidade total vendida para todo o país (geral e por região).