

Algoritmos e Linguagem de Programação - ALP

Rogério Eduardo da Silva - *rogerio.silva@udesc.br*

Adriano Fiorese - *adriano.fiorese@udesc.br*

Universidade do Estado de Santa Catarina
Departamento de Ciência da Computação

5 de maio de 2017

Conteúdo Programático:

Apresentação da Disciplina

Unidade 01 - Noções básicas sobre sistemas de computação (4 horas)

- Breve Histórico da Computação

- Conceitos de Informática

- Modelo de um Computador

Unidade 02 - Noções sobre linguagens de programação e programas (06 horas)

- Linguagens para Computadores

- Compiladores & Interpretadores

Unidade 03 - Estudo de uma linguagem de alto nível (28 horas)

- Algoritmos

- Tipos Primitivos de Dados

- Variáveis & Constantes

- Operadores & Expressões

- Comando de Atribuição

- Comandos de Entrada e Saída

- Processo de Compilação

- Interface ao Usuário

Comandos de Desvio de Fluxo de Execução: if-else
Comandos de Desvio de Fluxo de Execução: Repetições

Unidade 04 - Estruturas homogêneas e Sub-rotinas (26 horas)

Estruturas homogêneas

Vetores

Vetores Multi-dimensionais

Sub-Rotinas

Unidade 05 - Projeto Final (8 horas)

Ementa

- Noções básicas sobre sistemas de computação.
- Noções sobre linguagens de programação e programas.
- Estudo de uma linguagem de alto nível.

Objetivo Geral da Disciplina

Ao final do curso, o aluno deverá estar apto a:

- Dominar o processo de solução de problemas através do desenvolvimento de algoritmos a serem executados por computador.
- Dominar os comandos básicos, estruturar os dados em tipos simples e estruturados, utilizar conceitos de sub-programação, através da linguagem de programação C.

Objetivos Específicos

- CONCEITUAR os princípios básicos da computação.
- ENUMERAR os principais recursos de hardware disponíveis na ciência da computação.
- ENUMERAR os principais recursos de software disponíveis na ciência da computação.
- INTRODUZIR a lógica de programação.
- CARACTERIZAR as principais ferramentas auxiliares para programação.
- PROPORCIONAR práticas de programação.

Método de Ensino

- Aulas expositivas em sala e em laboratório
- Listas de exercícios teóricos e práticos
- Atendimento presencial (sala do professor) e/ou através da lista de emails da disciplina `alp-ee@googlegroups.com`
- `http://www.rogerioesilva.net/`

Avaliações

- Participação em Classe;
- Provas (2 provas previstas – 2 individuais e s/ consulta);
- Trabalhos individuais ou em grupos de 2 ou mais alunos, com o desenvolvimento de soluções para problemas sugeridos;

$$NotaFinal = Pr_1 * 0.3 + Pr_2 * 0.3 + TF * 0.3$$

- - Pr_1 Prova 1
 - Pr_2 Prova 2
 - TF Trabalho Final da Disciplina (Arduino)

Unidade 01

Noções básicas sobre sistemas de computação

Previsão: 04 horas/aula

Breve Histórico da Computação

- Necessidade de se realizar cálculos repetitivos
 - **COMPUTARE** = calcular

Breve Histórico da Computação

- Necessidade de se realizar cálculos repetitivos
 - **COMPUTARE = calcular**
- Primeiro dispositivo de cálculo: ábaco (3500 A.C.)
 - Realiza operações sobre uma representação no sistema decimal

Breve Histórico da Computação

- Necessidade de se realizar cálculos repetitivos
 - **COMPUTARE = calcular**
- Primeiro dispositivo de cálculo: ábaco (3500 A.C.)
 - Realiza operações sobre uma representação no sistema decimal
- (1550-1617) John Napier (inventor dos logaritmos naturais)
 - Dispositivo de bastões que continham números e era capaz de multiplicar e dividir automaticamente
 - Dispositivo com cartões chamado 'Estruturas de Napier' que fazia multiplicações

Breve Histórico da Computação

- Necessidade de se realizar cálculos repetitivos
 - **COMPUTARE = calcular**
- Primeiro dispositivo de cálculo: ábaco (3500 A.C.)
 - Realiza operações sobre uma representação no sistema decimal
- (1550-1617) John Napier (inventor dos logaritmos naturais)
 - Dispositivo de bastões que continham números e era capaz de multiplicar e dividir automaticamente
 - Dispositivo com cartões chamado 'Estruturas de Napier' que fazia multiplicações
- (1623-1662) Blaise Pascal
 - Primeira máquina automática de calcular ('Pascalina') = fazia adições e subtrações

Breve Histórico da Computação

- (1883) Charles Babbage
 - Projetou a “Máquina Analítica ou Diferencial”
 - Não chegou a ser construída mas previa programa, memória, unidade de controle e periféricos E/S
 - É considerado o pai da informática moderna

Breve Histórico da Computação

- (1883) Charles Babbage
 - Projetou a “Máquina Analítica ou Diferencial”
 - Não chegou a ser construída mas previa programa, memória, unidade de controle e periféricos E/S
 - É considerado o pai da informática moderna
- (1854) George Boole
 - Desenvolveu a **Álgebra de Boole** que permitiu mais tarde a criação da “Teoria dos Circuitos Lógicos”

Breve Histórico da Computação

- (1883) Charles Babbage
 - Projetou a “Máquina Analítica ou Diferencial”
 - Não chegou a ser construída mas previa programa, memória, unidade de controle e periféricos E/S
 - É considerado o pai da informática moderna
- (1854) George Boole
 - Desenvolveu a **Álgebra de Boole** que permitiu mais tarde a criação da “Teoria dos Circuitos Lógicos”
- (1937) Surge o primeiro computador eletromecânico: MARK-I
 - Somava dois números em menos de 1 segundo
 - Multiplicava dois números em 6 segundos

Breve Histórico da Computação

- (1883) Charles Babbage
 - Projetou a “Máquina Analítica ou Diferencial”
 - Não chegou a ser construída mas previa programa, memória, unidade de controle e periféricos E/S
 - É considerado o pai da informática moderna
- (1854) George Boole
 - Desenvolveu a **Álgebra de Boole** que permitiu mais tarde a criação da “Teoria dos Circuitos Lógicos”
- (1937) Surge o primeiro computador eletromecânico: MARK-I
 - Somava dois números em menos de 1 segundo
 - Multiplicava dois números em 6 segundos
- Em 1952 surgem os computadores MANIAC-I, MANIAC-II e UNIVAC-H

Breve Histórico da Computação

- (1883) Charles Babbage
 - Projetou a “Máquina Analítica ou Diferencial”
 - Não chegou a ser construída mas previa programa, memória, unidade de controle e periféricos E/S
 - É considerado o pai da informática moderna
- (1854) George Boole
 - Desenvolveu a **Álgebra de Boole** que permitiu mais tarde a criação da “Teoria dos Circuitos Lógicos”
- (1937) Surge o primeiro computador eletromecânico: MARK-I
 - Somava dois números em menos de 1 segundo
 - Multiplicava dois números em 6 segundos
- Em 1952 surgem os computadores MANIAC-I, MANIAC-II e UNIVAC-H
 - Surge a **Eletrônica**

Breve Histórico da Computação

A Eletrônica

- Nos anos 50 surge o diodo e o transistor

Breve Histórico da Computação

A Eletrônica

- Nos anos 50 surge o diodo e o transistor
 - Permitiram a miniaturização dos circuitos eletrônicos
 - Começa a era dos circuitos *Short Scale Integration - SSI*

Breve Histórico da Computação

A Eletrônica

- Nos anos 50 surge o diodo e o transistor
 - Permitiram a miniaturização dos circuitos eletrônicos
 - Começa a era dos circuitos *Short Scale Integration* - *SSI*
 - ... que logo se tornam *Medium Scale Integration* - *MSI* que continham de 100 a 1000 portas lógicas na mesma pastilha

Breve Histórico da Computação

A Eletrônica

- Nos anos 50 surge o diodo e o transistor
 - Permitiram a miniaturização dos circuitos eletrônicos
 - Começa a era dos circuitos *Short Scale Integration* - *SSI*
 - ... que logo se tornam *Medium Scale Integration* - *MSI* que continham de 100 a 1000 portas lógicas na mesma pastilha
 - Já os *Long Scale Integration* - *LSI* continham entre 1000 e 10000 portas lógicas

Breve Histórico da Computação

A Eletrônica

- Nos anos 50 surge o diodo e o transistor
 - Permitiram a miniaturização dos circuitos eletrônicos
 - Começa a era dos circuitos *Short Scale Integration* - *SSI*
 - ... que logo se tornam *Medium Scale Integration* - *MSI* que continham de 100 a 1000 portas lógicas na mesma pastilha
 - Já os *Long Scale Integration* - *LSI* continham entre 1000 e 10000 portas lógicas
 - Chegamos ao *Very Long Scale Integration* - *VLSI* ao se ultrapassar as 10000 portas lógicas

Breve Histórico da Computação

A Eletrônica

- Nos anos 50 surge o diodo e o transistor
 - Permitiram a miniaturização dos circuitos eletrônicos
 - Começa a era dos circuitos *Short Scale Integration - SSI*
 - ... que logo se tornam *Medium Scale Integration - MSI* que continham de 100 a 1000 portas lógicas na mesma pastilha
 - Já os *Long Scale Integration - LSI* continham entre 1000 e 10000 portas lógicas
 - Chegamos ao *Very Long Scale Integration - VLSI* ao se ultrapassar as 10000 portas lógicas
- (1971) Surge o microprocessador
 - Permitiu a implementação de toda a CPU em um único circuito integrado
 - Surgem os **computadores**

Breve Histórico da Computação

As gerações dos computadores

1ª geração a base de válvulas a vácuo; aplicações científicas e militares; utilizavam linguagem de máquina e cartões perfurados.

Breve Histórico da Computação

As gerações dos computadores

- 1ª geração a base de válvulas a vácuo; aplicações científicas e militares; utilizavam linguagem de máquina e cartões perfurados.
- 2ª geração a base de transistores; utilizavam linguagens de montagem (Assembly) e de mais alto nível como COBOL, ForTran e Algol. Usavam memórias magnéticas como fitas e tambores.

Breve Histórico da Computação

As gerações dos computadores

- 1ª geração a base de válvulas a vácuo; aplicações científicas e militares; utilizavam linguagem de máquina e cartões perfurados.
- 2ª geração a base de transistores; utilizavam linguagens de montagem (Assembly) e de mais alto nível como COBOL, ForTran e Algol. Usavam memórias magnéticas como fitas e tambores.
- 3ª geração a base de circuitos integrados (SSI e MSI); surgimento do *software* como sistemas operacionais; memórias a base de semicondutores e discos magnéticos.

Breve Histórico da Computação

As gerações dos computadores

- 1ª geração a base de válvulas a vácuo; aplicações científicas e militares; utilizavam linguagem de máquina e cartões perfurados.
- 2ª geração a base de transistores; utilizavam linguagens de montagem (Assembly) e de mais alto nível como COBOL, ForTran e Algol. Usavam memórias magnéticas como fitas e tambores.
- 3ª geração a base de circuitos integrados (SSI e MSI); surgimento do *software* como sistemas operacionais; memórias a base de semicondutores e discos magnéticos.
- 4ª geração advento do microprocessador; usa LSI; armazenada em *Floppy disks*; uso das linguagens de programação e o surgimento das redes de comunicação de dados.

Breve Histórico da Computação

As gerações dos computadores

- 1ª geração a base de válvulas a vácuo; aplicações científicas e militares; utilizavam linguagem de máquina e cartões perfurados.
- 2ª geração a base de transistores; utilizavam linguagens de montagem (Assembly) e de mais alto nível como COBOL, ForTran e Algol. Usavam memórias magnéticas como fitas e tambores.
- 3ª geração a base de circuitos integrados (SSI e MSI); surgimento do *software* como sistemas operacionais; memórias a base de semicondutores e discos magnéticos.
- 4ª geração advento do microprocessador; usa LSI; armazena em *Floppy disks*; uso das linguagens de programação e o surgimento das redes de comunicação de dados.
- 5ª geração ainda teórica; utilizaria inteligência artificial, linguagem natural, altíssima capacidade de processamento através de processadores ópticos ou quânticos.

Conceitos de Informática

- Uma definição é a **“ciência que estuda o tratamento automático e racional da informação”**

Conceitos de Informática

- Uma definição é a “**ciência que estuda o tratamento automático e racional da informação**”
- O termo surgiu na França (1962) da junção das palavras **Information** *automatique*

Conceitos de Informática

- Uma definição é a “**ciência que estuda o tratamento automático e racional da informação**”
- O termo surgiu na França (1962) da junção das palavras **Information** *automatique*
- Principais funções
 - desenvolvimento de novas máquinas
 - desenvolvimento de novos métodos de trabalho
 - construção de aplicações automáticas
 - melhoria de métodos e aplicações existentes

Conceitos de Informática

- Uma definição é a **“ciência que estuda o tratamento automático e racional da informação”**

Conceitos de Informática

- Uma definição é a “**ciência que estuda o tratamento automático e racional da informação**”
- O termo surgiu na França (1962) da junção das palavras **Information** *automatique*

Conceitos de Informática

- Uma definição é a “**ciência que estuda o tratamento automático e racional da informação**”
- O termo surgiu na França (1962) da junção das palavras **Information** *automatique*
- Principais funções
 - desenvolvimento de novas máquinas
 - desenvolvimento de novos métodos de trabalho
 - construção de aplicações automáticas
 - melhoria de métodos e aplicações existentes

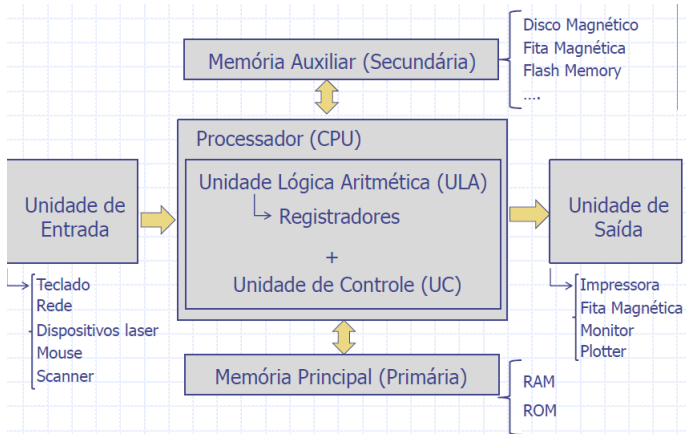
Modelo de um Computador

- Computador é uma máquina composta de elementos físicos do tipo eletrônico, que executa instruções com alta velocidade e precisão desde que corretamente instruído.

Modelo de um Computador

- Computador é uma máquina composta de elementos físicos do tipo eletrônico, que executa instruções com alta velocidade e precisão desde que corretamente instruído.
- **Hardware** conjunto de todos os componentes físicos da máquina (teclado, mouse, monitor, impressora, placa mãe, etc.)
- **Software** conjunto dos componentes lógicos que são executados pelo hardware e servem para controlá-lo

Modelo de um Computador



Modelo de um Computador

A CPU

Unidade Lógica Aritmética (ULA) responsável pelas operações elementares: aritméticas e lógicas.

Modelo de um Computador

A CPU

Unidade Lógica Aritmética (ULA) responsável pelas operações elementares: aritméticas e lógicas.

Registradores unidades de memória RAM para execução de operações pela ULA e UC.

Modelo de um Computador

A CPU

Unidade Lógica Aritmética (ULA) responsável pelas operações elementares: aritméticas e lógicas.

Registradores unidades de memória RAM para execução de operações pela ULA e UC.

Unidade de Controle (UC) controla o fluxo de dados entre as unidades da CPU, buscando as operações na memória principal e distribuindo entre os módulos responsáveis pela execução (ULA, E/S, etc.)

Modelo de um Computador

A Memória

Memória Primária ou Principal local de armazenamento das instruções e dados durante a execução dos programas.

Modelo de um Computador

A Memória

Memória Primária ou Principal local de armazenamento das instruções e dados durante a execução dos programas.

Random Access Memory (RAM) É uma memória volátil (depende da máquina estar ligada). Tempo de leitura/gravação rápidos. Acesso aleatório

Modelo de um Computador

A Memória

Memória Primária ou Principal local de armazenamento das instruções e dados durante a execução dos programas.

Random Access Memory (RAM) É uma memória volátil (depende da máquina estar ligada). Tempo de leitura/gravação rápidos. Acesso aleatório

Read Only Memory (ROM) armazena um conjunto de instruções do fabricante utilizadas durante o processo de inicialização do computador. Dados somente para leitura.

Modelo de um Computador

A Memória

Memória Primária ou Principal local de armazenamento das instruções e dados durante a execução dos programas.

Random Access Memory (RAM) É uma memória volátil (depende da máquina estar ligada). Tempo de leitura/gravação rápidos. Acesso aleatório

Read Only Memory (ROM) armazena um conjunto de instruções do fabricante utilizadas durante o processo de inicialização do computador. Dados somente para leitura.

Modelo de um Computador

A Memória

Memória Secundária ou Auxiliar conjunto dos dispositivos periféricos de armazenamento permanente de dados.

Modelo de um Computador

A Memória

Memória Secundária ou Auxiliar conjunto dos dispositivos periféricos de armazenamento permanente de dados.

Meio Magnético utiliza uma camada de óxido de ferro para registrar informações em pontos magnetizáveis.
Ex.: Discos e fitas magnéticas.

Modelo de um Computador

A Memória

Memória Secundária ou Auxiliar conjunto dos dispositivos periféricos de armazenamento permanente de dados.

Meio Magnético utiliza uma camada de óxido de ferro para registrar informações em pontos magnetizáveis.
Ex.: Discos e fitas magnéticas.

Meio Óptico efetuar marcações a laser em uma superfície plástica reativa. Ex.: CD, DVD, Blu-Ray.

Modelo de um Computador

A Memória

Memória Secundária ou Auxiliar conjunto dos dispositivos periféricos de armazenamento permanente de dados.

Meio Magnético utiliza uma camada de óxido de ferro para registrar informações em pontos magnetizáveis.
Ex.: Discos e fitas magnéticas.

Meio Óptico efetuar marcações a laser em uma superfície plástica reativa. Ex.: CD, DVD, Blu-Ray.

Modelo de um Computador

Os periféricos

- Qualquer dispositivo que permite a comunicação entre o computador e o mundo exterior.

Modelo de um Computador

Os periféricos

- Qualquer dispositivo que permite a comunicação entre o computador e o mundo exterior.
- Esta comunicação (transferência de dados) pode ser realizada em blocos ou sequencialmente (palavra por palavra).

Modelo de um Computador

Os periféricos

- Qualquer dispositivo que permite a comunicação entre o computador e o mundo exterior.
- Esta comunicação (transferência de dados) pode ser realizada em blocos ou sequencialmente (palavra por palavra).

Dispositivos de Entrada qualquer dispositivo capaz de enviar informações do mundo exterior para o computador. Ex.: teclado, mouse, scanner, leitor de códigos de barra, sensores, etc.

Modelo de um Computador

Os periféricos

- Qualquer dispositivo que permite a comunicação entre o computador e o mundo exterior.
- Esta comunicação (transferência de dados) pode ser realizada em blocos ou sequencialmente (palavra por palavra).

Dispositivos de Entrada qualquer dispositivo capaz de enviar informações do mundo exterior para o computador. Ex.: teclado, mouse, scanner, leitor de códigos de barra, sensores, etc.

Dispositivos de Saída qualquer dispositivo capaz de converter informações do computador para uma forma inteligível e enviar para o mundo exterior. Exemplo: monitor, impressora, plotter, etc.

Modelo de um Computador

Os periféricos

- Qualquer dispositivo que permite a comunicação entre o computador e o mundo exterior.
- Esta comunicação (transferência de dados) pode ser realizada em blocos ou sequencialmente (palavra por palavra).

Dispositivos de Entrada qualquer dispositivo capaz de enviar informações do mundo exterior para o computador. Ex.: teclado, mouse, scanner, leitor de códigos de barra, sensores, etc.

Dispositivos de Saída qualquer dispositivo capaz de converter informações do computador para uma forma inteligível e enviar para o mundo exterior. Exemplo: monitor, impressora, plotter, etc.

- **Pergunta: memória auxiliar pode ser considerada dispositivo de saída?**

Software

- É a abstração lógica composta de um conjunto de instruções, organizadas e armazenadas em um ou mais arquivos, que instruem o computador a executar tarefas que solucionam determinados problemas.

Software

- É a abstração lógica composta de um conjunto de instruções, organizadas e armazenadas em um ou mais arquivos, que instruem o computador a executar tarefas que solucionam determinados problemas.

Básico (sistema) responsáveis por administrar, operar e manter o funcionamento do computador. É o ambiente onde os demais softwares são executados. Ex.: sistemas operacionais.

Software

- É a abstração lógica composta de um conjunto de instruções, organizadas e armazenadas em um ou mais arquivos, que instruem o computador a executar tarefas que solucionam determinados problemas.

Básico (sistema) responsáveis por administrar, operar e manter o funcionamento do computador. É o ambiente onde os demais softwares são executados. Ex.: sistemas operacionais.

Aplicação responsáveis pela execução de tarefas através do uso do computador. Ex.: processador de texto e gráficos, planilhas eletrônicas, jogos, gerenciados de banco de dados, etc.

Software

- É a abstração lógica composta de um conjunto de instruções, organizadas e armazenadas em um ou mais arquivos, que instruem o computador a executar tarefas que solucionam determinados problemas.

Básico (sistema) responsáveis por administrar, operar e manter o funcionamento do computador. É o ambiente onde os demais softwares são executados. Ex.: sistemas operacionais.

Aplicação responsáveis pela execução de tarefas através do uso do computador. Ex.: processador de texto e gráficos, planilhas eletrônicas, jogos, gerenciados de banco de dados, etc.

Utilitário software de apoio à operação do computador. Executa rotinas auxiliares frequentes como: (des)compactação, detecção/eliminação de vírus, etc.

Unidade 02

Noções sobre linguagens de programação e programas

Previsão: 06 horas/aula

Linguagens para Computadores

Linguagem Binária

- Dispositivos eletrônicos que compõe o computador distinguem apenas 2 sinais elétricos denominados **bit**.
 - Presença de sinal elétrico representado pelo símbolo 1.
 - Ausência de sinal elétrico representado pelo símbolo 0.

Linguagens para Computadores

Linguagem Binária

- Dispositivos eletrônicos que compõe o computador distinguem apenas 2 sinais elétricos denominados **bit**.
 - Presença de sinal elétrico representado pelo símbolo 1.
 - Ausência de sinal elétrico representado pelo símbolo 0.
- Uma sequência de bits pode codificar dados ou instruções que a CPU é capaz de executar.

101011110001101010

Linguagens para Computadores

Linguagem de Máquina

- Descreve a linguagem constituída pelas instruções que podem ser diretamente executadas pela CPU.
 - Somar, carregar valores, comparar valores, movimentar valores na memória, desviar a execução para uma instrução específica.
 - Cada instrução é representada por uma determinada sequência binária.

Linguagens para Computadores

Linguagem de Máquina

- Descreve a linguagem constituída pelas instruções que podem ser diretamente executadas pela CPU.
 - Somar, carregar valores, comparar valores, movimentar valores na memória, desviar a execução para uma instrução específica.
 - Cada instrução é representada por uma determinada sequência binária.
- Inicialmente, codificar manualmente uma sequência binária era a única forma que os programadores dispunham para desenvolver seus programas.

Linguagens para Computadores

Linguagem de Máquina

- Descreve a linguagem constituída pelas instruções que podem ser diretamente executadas pela CPU.
 - Somar, carregar valores, comparar valores, movimentar valores na memória, desviar a execução para uma instrução específica.
 - Cada instrução é representada por uma determinada sequência binária.
- Inicialmente, codificar manualmente uma sequência binária era a única forma que os programadores dispunham para desenvolver seus programas.
- Posteriormente, foi introduzido o conceito de “mnemônicos” que nada mais são do que ‘apelidos’ para determinadas sequências binárias, a fim de facilitar sua programação:

Linguagens para Computadores

Linguagem de Máquina

- Descreve a linguagem constituída pelas instruções que podem ser diretamente executadas pela CPU.
 - Somar, carregar valores, comparar valores, movimentar valores na memória, desviar a execução para uma instrução específica.
 - Cada instrução é representada por uma determinada sequência binária.
- Inicialmente, codificar manualmente uma sequência binária era a única forma que os programadores dispunham para desenvolver seus programas.
- Posteriormente, foi introduzido o conceito de “mnemônicos” que nada mais são do que ‘apelidos’ para determinadas sequências binárias, a fim de facilitar sua programação:

```
MOV R1, x
ADD R1, R2
JMP L
```

Linguagens para Computadores

Linguagem de Máquina

- Descreve a linguagem constituída pelas instruções que podem ser diretamente executadas pela CPU.
 - Somar, carregar valores, comparar valores, movimentar valores na memória, desviar a execução para uma instrução específica.
 - Cada instrução é representada por uma determinada sequência binária.
- Inicialmente, codificar manualmente uma sequência binária era a única forma que os programadores dispunham para desenvolver seus programas.
- Posteriormente, foi introduzido o conceito de “mnemônicos” que nada mais são do que ‘apelidos’ para determinadas sequências binárias, a fim de facilitar sua programação:

```
MOV R1, x
ADD R1, R2
JMP L
```

- A evolução seguinte foi automatizar o processo de tradução de mnemônicos em linguagem de máquina (denominado ‘montagem’) através de *programas montadores*.

Linguagens para Computadores

Linguagens de Médio e Alto Nível

- Descrevem linguagens constituídas por um conjunto mais rico de operações e construções sintáticas.

Linguagens para Computadores

Linguagens de Médio e Alto Nível

- Descrevem linguagens constituídas por um conjunto mais rico de operações e construções sintáticas.
- De acordo com o nível de abstração exigido para o desenvolvimento de um programa é que se classifica a complexidade da linguagem

Linguagens para Computadores

Linguagens de Médio e Alto Nível

- Descrevem linguagens constituídas por um conjunto mais rico de operações e construções sintáticas.
- De acordo com o nível de abstração exigido para o desenvolvimento de um programa é que se classifica a complexidade da linguagem
 - Baixo Nível exigem um grande conhecimento do funcionamento do hardware.
Exemplos: ling. de máquina e de montagem (Assembly)

Linguagens para Computadores

Linguagens de Médio e Alto Nível

- Descrevem linguagens constituídas por um conjunto mais rico de operações e construções sintáticas.
- De acordo com o nível de abstração exigido para o desenvolvimento de um programa é que se classifica a complexidade da linguagem
 - Baixo Nível exigem um grande conhecimento do funcionamento do hardware. Exemplos: ling. de máquina e de montagem (Assembly)
 - Médio Nível introduziu o conceito de comandos (*statements*) porém ainda exige um bom conhecimento em termos de lógica permitindo uma abstração maior acerca dos recursos de hardware.

Linguagens para Computadores

Linguagens de Médio e Alto Nível

- Descrevem linguagens constituídas por um conjunto mais rico de operações e construções sintáticas.
- De acordo com o nível de abstração exigido para o desenvolvimento de um programa é que se classifica a complexidade da linguagem
 - Baixo Nível exigem um grande conhecimento do funcionamento do hardware. Exemplos: ling. de máquina e de montagem (Assembly)
 - Médio Nível introduziu o conceito de comandos (*statements*) porém ainda exige um bom conhecimento em termos de lógica permitindo uma abstração maior acerca dos recursos de hardware.
 - Alto Nível permite ao programador focar apenas no processo lógico do algoritmos, abstraindo completamente questão relacionadas ao hardware no qual o programa será executado.

Linguagens para Computadores

Linguagens de Médio e Alto Nível

- Descrevem linguagens constituídas por um conjunto mais rico de operações e construções sintáticas.
- De acordo com o nível de abstração exigido para o desenvolvimento de um programa é que se classifica a complexidade da linguagem
 - Baixo Nível exigem um grande conhecimento do funcionamento do hardware. Exemplos: ling. de máquina e de montagem (Assembly)
 - Médio Nível introduziu o conceito de comandos (*statements*) porém ainda exige um bom conhecimento em termos de lógica permitindo uma abstração maior acerca dos recursos de hardware.
 - Alto Nível permite ao programador focar apenas no processo lógico do algoritmos, abstraindo completamente questão relacionadas ao hardware no qual o programa será executado.
Um programa escrito em linguagem de alto nível precisa ser traduzido para linguagem de máquina antes que possa ser executado. Isso é feito através de um processo denominado **compilação**.

Compiladores & Interpretadores

Compilador

- É um programa tradutor de programas escritos em um determinada linguagem (**linguagem fonte**) para outra equivalente (**linguagem objeto**).

Compiladores & Interpretadores

Compilador

- É um programa tradutor de programas escritos em um determinada linguagem (**linguagem fonte**) para outra equivalente (**linguagem objeto**).
- As linguagens modernas atualmente permitem a utilização de códigos-fonte auxiliares pré-compilados (*bibliotecas*) conjuntamente com os programas desenvolvidos pelo programador. Para tal, é necessário que estas bibliotecas sejam combinadas ao código-objeto do programa através de um processo denominado **linkedição**.

Compiladores & Interpretadores

Compilador

- É um programa tradutor de programas escritos em um determinada linguagem (**linguagem fonte**) para outra equivalente (**linguagem objeto**).
- As linguagens modernas atualmente permitem a utilização de códigos-fonte auxiliares pré-compilados (*bibliotecas*) conjuntamente com os programas desenvolvidos pelo programador. Para tal, é necessário que estas bibliotecas sejam combinadas ao código-objeto do programa através de um processo denominado **linkedição**.
- Uma outra atribuição dos compiladores é a análise do código-fonte. Isto significa que antes de iniciar a tradução do código-fonte, o mesmo é verificado se as instruções nele contidas respeitam as regras pré-estabelecidas pela linguagem em questão; caso contrário, uma mensagem de erro é enviada ao programador, informando sobre o erro.

Compiladores & Interpretadores

Interpretador

- É um programa que executa diretamente as instruções escritas em um determinada linguagem fonte.

Compiladores & Interpretadores

Interpretador

- É um programa que executa diretamente as instruções escritas em um determinada linguagem fonte.
- A execução de programas através de interpretação é um processo mais lento que a compilação dado que exige a utilização de uma 'máquina interpretadora'.

Compiladores & Interpretadores

Interpretador

- É um programa que executa diretamente as instruções escritas em um determinada linguagem fonte.
- A execução de programas através de interpretação é um processo mais lento que a compilação dado que exige a utilização de uma 'máquina interpretadora'.
- Um mesmo código fonte pode ser interpretado em diferentes plataformas (Sistemas Operacionais) desde que hajam máquinas interpretadores desenvolvidas para aquela plataforma específica. Ex.: JAVA.

Compiladores & Interpretadores

Interpretador

- É um programa que executa diretamente as instruções escritas em um determinada linguagem fonte.
- A execução de programas através de interpretação é um processo mais lento que a compilação dado que exige a utilização de uma 'máquina interpretadora'.
- Um mesmo código fonte pode ser interpretado em diferentes plataformas (Sistemas Operacionais) desde que hajam máquinas interpretadores desenvolvidas para aquela plataforma específica. Ex.: JAVA.
- Um programa compilado só poderá ser executado **após** o código-objeto ser gerado (não apresentar erros de sintaxe). Um programa interpretado é sempre executado e encerra a execução quando encontra o primeiro erro de sintaxe (*Abort execution*).

Ambiente de Programação

Editor de Texto permite editar qualquer arquivo em formato texto inclusive programas fonte.

Ambiente de Programação

Editor de Texto permite editar qualquer arquivo em formato texto inclusive programas fonte.

- Ex. de editores de texto: Bloco de Notas, Sublime, Geany,

Ambiente de Programação

Editor de Texto permite editar qualquer arquivo em formato texto inclusive programas fonte.

- Ex. de editores de texto: Bloco de Notas, Sublime, Geany,

Ambiente de Desenvolvimento Integrado (*Integrated Development Environment - IDE*) é um editor especial dedicado à edição de programas fonte, inclui diversos recursos que aceleram/facilitam a edição de programas fonte: ênfase de sintaxe, auto-completar, ajuda online, compilação/linkedição integrada, entre outros.

Ambiente de Programação

Editor de Texto permite editar qualquer arquivo em formato texto inclusive programas fonte.

- Ex. de editores de texto: Bloco de Notas, Sublime, Geany,

Ambiente de Desenvolvimento Integrado (*Integrated Development Environment - IDE*) é um editor especial dedicado à edição de programas fonte, inclui diversos recursos que aceleram/facilitam a edição de programas fonte: ênfase de sintaxe, auto-completar, ajuda online, compilação/linkedição integrada, entre outros.

- Ex. de IDE: MSVC, Dev-C++, CodeBlocks, Eclipse,

Ambiente de Programação

Editor de Texto permite editar qualquer arquivo em formato texto inclusive programas fonte.

- Ex. de editores de texto: Bloco de Notas, Sublime, Geany,

Ambiente de Desenvolvimento Integrado (*Integrated Development Environment - IDE*) é um editor especial dedicado à edição de programas fonte, inclui diversos recursos que aceleram/facilitam a edição de programas fonte: ênfase de sintaxe, auto-completar, ajuda online, compilação/linkedição integrada, entre outros.

- Ex. de IDE: MSVC, Dev-C++, CodeBlocks, Eclipse,

Depurador programa auxiliar que permite o acompanhamento da execução de um programa. Tem o objetivo de auxiliar o programador na detecção/correção de erros/melhorias no programa.

Ambiente de Programação

Editor de Texto permite editar qualquer arquivo em formato texto inclusive programas fonte.

- Ex. de editores de texto: Bloco de Notas, Sublime, Geany,

Ambiente de Desenvolvimento Integrado (*Integrated Development Environment - IDE*) é um editor especial dedicado à edição de programas fonte, inclui diversos recursos que aceleram/facilitam a edição de programas fonte: ênfase de sintaxe, auto-completar, ajuda online, compilação/linkedição integrada, entre outros.

- Ex. de IDE: MSVC, Dev-C++, CodeBlocks, Eclipse,

Depurador programa auxiliar que permite o acompanhamento da execução de um programa. Tem o objetivo de auxiliar o programador na detecção/correção de erros/melhorias no programa.

Unidade 03

Estudo de uma linguagem de alto nível

Previsão: 28 horas/aula

Algoritmos

- *“Consiste em uma sequência finita de regras ou instruções que especificam como determinadas operações básicas, executáveis automaticamente, devem ser combinadas para a realização de uma tarefa desejada”.*

Algoritmos

- *“Consiste em uma sequência finita de regras ou instruções que especificam como determinadas operações básicas, executáveis automaticamente, devem ser combinadas para a realização de uma tarefa desejada”.*
- *“Descrição de um comportamento expresso em termos de um repertório bem sucedido e finito de ações naturais, das quais damos por certo que elas podem ser executadas para resolver um problema.”*

Algoritmos vs. Programas

- Algoritmos são descrições de processos (modelo abstrato).

Algoritmos vs. Programas

- Algoritmos são descrições de processos (modelo abstrato).
- Programas são a realização dos processos descritos (modelo concreto).

Algoritmos vs. Programas

- Algoritmos são descrições de processos (modelo abstrato).
- Programas são a realização dos processos descritos (modelo concreto).
- Exemplos práticos:
 - Receita Culinária vs Cozinhar
 - Partitura Musical vs Tocar um instrumento
 - Projetar um artefato vs Construir/utilizar um artefato

Algoritmos vs. Programas

- Algoritmos são descrições de processos (modelo abstrato).
- Programas são a realização dos processos descritos (modelo concreto).
- Exemplos práticos:
 - Receita Culinária vs Cozinhar
 - Partitura Musical vs Tocar um instrumento
 - Projetar um artefato vs Construir/utilizar um artefato
- Existem diversas formas de se descrever algoritmos.

Algoritmos - Formas de Representação

Sistemático Descritivo ou Narração Descritiva usa a linguagem natural (p.ex.: língua Portuguesa) para descrever um procedimento

Receita de Bolo

1. Prepare uma certa lista de ingredientes
2. Misture os ingredientes
3. Despeje a mistura numa forma
4. Se tiver côco ralado então
 - 4.1 Adicione côco ralado
5. Ligue o forno a 200 graus Celsius
6. Leve a forma ao forno
7. Enquanto não estiver assado
 - 7.1 Deixe a forma no forno
8. Retire do forno

Algoritmos - Formas de Representação

Sistemático Descritivo ou Narração Descritiva usa a linguagem natural (p.ex.: língua Portuguesa) para descrever um procedimento

Receita de Bolo

1. Prepare uma certa lista de ingredientes
2. Misture os ingredientes
3. Despeje a mistura numa forma
4. Se tiver côco ralado então
 - 4.1 Adicione côco ralado
5. Ligue o forno a 200 graus Celsius
6. Leve a forma ao forno
7. Enquanto não estiver assado
 - 7.1 Deixe a forma no forno
8. Retire do forno

Como Trocar um Pneu Furado

1. Afrouxar ligeiramente as porcas
2. Suspender o carro
3. Retirar todas as porcas e o pneu
4. Colocar o pneu estepe e recolocar as porcas
5. Apertar as porcas
6. Abaixar o carro
7. Se as porcas tiverem alguma folga
 - 7.1 Apertar as porcas até seu perfeito encaixe

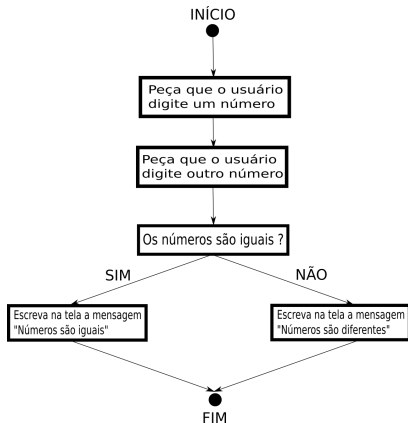
Algoritmos - Formas de Representação

Exercício

- Você foi encarregado de ensinar um aluno novo recém chegado a Joinville como ele deve fazer para ir do Campus da UDESC até o Shopping Mueller.
 - Detalhe: Você não sabe se o aluno em questão tem ou não carro, portanto considere ambas as possibilidades

Algoritmos - Formas de Representação

Fluxograma



Algoritmos - Formas de Representação

FLUXOGRAMA



Sentido do fluxo
de dados



Desvio da execução
do programa



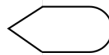
Terminal de INÍCIO ou
FIM do processamento



Entrada manual
de dados



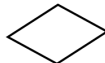
Processamento
em geral



Exibe informações
de saída



E/S em dispositivo
genérico (ex: arquivo)



Tomada de decisão



Conector de página

(caso o fluxograma seja muito complexo)

Algoritmos - Formas de Representação

Exercício

- Refazer o algoritmo do exercício anterior (caminho UDESC → Shopping Mueller) através de fluxogramas

Algoritmos - Formas de Representação

Pseudocódigo ou Sistemático Descritivo Padronizado usa um versão padronizada da linguagem natural

Algoritmo <nome do algoritmo>

 <área de declarações iniciais>

 <área de sub-rotinas>

Início

 <corpo do algoritmo>

Fim

Iniciação aos Algoritmos

Tipos Primitivos de Dados

- Classifica os dados utilizados por um programa de acordo o tipo da informação representada.

Iniciação aos Algoritmos

Tipos Primitivos de Dados

- Classifica os dados utilizados por um programa de acordo o tipo da informação representada.
- Informa ao compilador a quantidade de memória que precisa ser reservada para o armazenamento do dado específico.

Iniciação aos Algoritmos

Dados Numéricos

Inteiro (int) representa qualquer número (positivo ou negativo) que não possui parte fracionária. Ex: 0, 1, 1000, -125

Iniciação aos Algoritmos

Dados Numéricos

Inteiro (int) representa qualquer número (positivo ou negativo) que não possui parte fracionária. Ex: 0, 1, 1000, -125

Real (float) representa números (+/-) com sua parte fracionária. Também são chamados de números com 'ponto flutuante'.
Ex: 3.14159, 0.0, -124.77

Iniciação aos Algoritmos

Dados Literais

Character (char) representa qualquer caracter da tabela ASCII. Ex: letras, dígitos (não são números), espaço em branco, sinais ortográficos ou aritméticos, etc.
Um caracter é sempre descrito entre aspas simples como 'A' ou '5' ou ainda '('.

Iniciação aos Algoritmos

Dados Literais

- Caracter (char)** representa qualquer caracter da tabela ASCII. Ex: letras, dígitos (não são números), espaço em branco, sinais ortográficos ou aritméticos, etc.
Um caracter é sempre descrito entre aspas simples como 'A' ou '5' ou ainda '('.
- Cadeia (char [])** representa uma sequência de caracteres que permitem descrever palavras ou frases inteiras. Comumente é referido pelo termo em inglês **string**.
Uma *string* é representada por aspas inglesas "UDESC" ou "Semestre 2016/1"

Iniciação aos Algoritmos

Lógico (bool ou int) também chamado de **booleano**, são valores que armazenam apenas os valores lógico **TRUE** ou **FALSE**

Iniciação aos Algoritmos

Lógico (bool ou int) também chamado de **booleano**, são valores que armazenam apenas os valores lógico **TRUE** ou **FALSE**. Tradicionalmente, são utilizados para armazenar o resultado de *expressões lógicas* (comparações).

Iniciação aos Algoritmos

Exercício

- Classificar cada dado abaixo como (R) = real, (I) inteiro, (L) literal, (C) caracter ou (B) lógico:

<input type="checkbox"/> 1	<input type="checkbox"/> true	<input type="checkbox"/> "abC"	<input type="checkbox"/> -1
<input type="checkbox"/> 1.0	<input type="checkbox"/> "0.0"	<input type="checkbox"/> +32	<input type="checkbox"/> 'a'
<input type="checkbox"/> '1'	<input type="checkbox"/> "false"	<input type="checkbox"/> false	<input type="checkbox"/> "'a'"

Iniciação aos Algoritmos

Variáveis & Constantes

- Para que se possa armazenar e manipular dados em um programa, é necessário a pré-reserva de um espaço na memória do computador para este fim. Este processo é denominado *alocação de memória*.

Iniciação aos Algoritmos

Variáveis & Constantes

- Para que se possa armazenar e manipular dados em um programa, é necessário a pré-reserva de um espaço na memória do computador para este fim. Este processo é denominado *alocação de memória*.
- Cada um desses espaços pré-reservados é denominado de **variável** ou **constante**. Uma clara alusão ao conceito matemático que é uma abstração a uma informação manipulável.

Iniciação aos Algoritmos

Variáveis & Constantes

- Para que se possa armazenar e manipular dados em um programa, é necessário a pré-reserva de um espaço na memória do computador para este fim. Este processo é denominado *alocação de memória*.
- Cada um desses espaços pré-reservados é denominado de **variável** ou **constante**. Uma clara alusão ao conceito matemático que é uma abstração a uma informação manipulável.
- Tanto as variáveis quanto as constantes são declaradas da mesma forma:
Nome do identificador sequência de 1 a 32 letras, dígitos ou sinal de “_” não iniciada por dígito que identifica a memória de maneira única. Não é permitido o uso de palavras reservadas da própria linguagem como identificadores.

Iniciação aos Algoritmos

Variáveis & Constantes

- Para que se possa armazenar e manipular dados em um programa, é necessário a pré-reserva de um espaço na memória do computador para este fim. Este processo é denominado *alocação de memória*.
- Cada um desses espaços pré-reservados é denominado de **variável** ou **constante**. Uma clara alusão ao conceito matemático que é uma abstração a uma informação manipulável.
- Tanto as variáveis quanto as constantes são declaradas da mesma forma:
 - Nome do identificador sequência de 1 a 32 letras, dígitos ou sinal de “_” não iniciada por dígito que identifica a memória de maneira única. Não é permitido o uso de palavras reservadas da própria linguagem como identificadores.
 - Tipo de dado tipo da informação que será armazenada no referido espaço de memória. Indica para o compilador a quantidade de memória (em bytes) que será necessária ser alocada;

Iniciação aos Algoritmos

Variáveis & Constantes

- Para que se possa armazenar e manipular dados em um programa, é necessário a pré-reserva de um espaço na memória do computador para este fim. Este processo é denominado *alocação de memória*.
- Cada um desses espaços pré-reservados é denominado de **variável** ou **constante**. Uma clara alusão ao conceito matemático que é uma abstração a uma informação manipulável.
- Tanto as variáveis quanto as constantes são declaradas da mesma forma:
 - Nome do identificador sequência de 1 a 32 letras, dígitos ou sinal de “_” não iniciada por dígito que identifica a memória de maneira única. Não é permitido o uso de palavras reservadas da própria linguagem como identificadores.
 - Tipo de dado tipo da informação que será armazenada no referido espaço de memória. Indica para o compilador a quantidade de memória (em bytes) que será necessária ser alocada;
 - Endereço de memória posição da memória onde a variável/constante foi armazenada;

Iniciação aos Algoritmos

Variáveis & Constantes

- Para que se possa armazenar e manipular dados em um programa, é necessário a pré-reserva de um espaço na memória do computador para este fim. Este processo é denominado *alocação de memória*.
- Cada um desses espaços pré-reservados é denominado de **variável** ou **constante**. Uma clara alusão ao conceito matemático que é uma abstração a uma informação manipulável.
- Tanto as variáveis quanto as constantes são declaradas da mesma forma:
 - Nome do identificador** sequência de 1 a 32 letras, dígitos ou sinal de “_” não iniciada por dígito que identifica a memória de maneira única. Não é permitido o uso de palavras reservadas da própria linguagem como identificadores.
 - Tipo de dado** tipo da informação que será armazenada no referido espaço de memória. Indica para o compilador a quantidade de memória (em bytes) que será necessária ser alocada;
 - Endereço de memória** posição da memória onde a variável/constante foi armazenada;
 - Informação armazenada** o valor armazenado no referido espaço de memória. No caso das variáveis, esse valor pode ser alterado no tempo; já para as constantes não.

Iniciação aos Algoritmos

Variáveis & Constantes

- Exemplos de identificadores válidos: X, soma, Nota1, _ALP, Media_da_Turma_310

Iniciação aos Algoritmos

Variáveis & Constantes

- Exemplos de identificadores válidos: X, soma, Nota1, _ALP, Media_da_Turma_310
- Não são aceitos os caracteres de acentos da lingua portuguesa como ã, é, ç, etc.

Iniciação aos Algoritmos

Variáveis & Constantes

- Exemplos de identificadores válidos: X, soma, Nota1, _ALP, Media_da_Turma_310
- Não são aceitos os caracteres de acentos da lingua portuguesa como ã, é, ç, etc.
- A linguagem C é do tipo *caso sensitivo*, o que significa que ela diferencia letras maiúsculas de minúsculas e portanto Teste \neq teste

Iniciação aos Algoritmos

Variáveis & Constantes

- Exemplos de identificadores válidos: X, soma, Nota1, _ALP, Media_da_Turma_310
- Não são aceitos os caracteres de acentos da lingua portuguesa como ã, é, ç, etc.
- A linguagem C é do tipo *caso sensetivo*, o que significa que ela diferencia letras maiúsculas de minúsculas e portanto Teste \neq teste
- Indique quais dos exemplos abaixo são válidos como nome de identificadores:
 1. X_1_Y_2
 2. _b
 3. 123XYZ
 4. _
 5. X1234567890
 6. Salário
 7. X e Y
 8. Km/h
 9. A-B

Iniciação aos Algoritmos

Variáveis & Constantes

- O comando para declaração de variáveis em linguagem C é dado por:

`<tipo de dados> <identificador> ;`

ou

`<tipo de dados> <identificador> = <valor> ;`

Iniciação aos Algoritmos

Variáveis & Constantes

- O comando para declaração de variáveis em linguagem C é dado por:
`<tipo de dados> <identificador> ;`

ou

`<tipo de dados> <identificador> = <valor> ;`

- O comando para declaração de variáveis em linguagem C é dado por:
`const <tipo de dados> <identificador> = <valor> ;`

ou

`#define <identificador> <valor>`

Iniciação aos Algoritmos

Variáveis & Constantes

- Exemplos válidos de declaração de variáveis:

```
int x;  
float PI = 3.14159;  
char sexo = 'M';  
char nome[10];  
bool resultado = false;
```

Iniciação aos Algoritmos

Variáveis & Constantes

- Exemplos válidos de declaração de variáveis:

```
int x;
float PI = 3.14159;
char sexo = 'M';
char nome[10];
bool resultado = false;
```

- Exemplos válidos de declaração de constantes:

```
#define PI 3.14159
const int TOTAL_DE_ALUNOS = 25;
```

Iniciação aos Algoritmos

Operadores

Operadores Aritméticos realizam operações aritméticas básicas: adição (+), subtração (-), multiplicação (*), divisão (/) [inteira e real], inversão de sinal (-) e resto da divisão (%).

Iniciação aos Algoritmos

Operadores

Operadores Aritméticos realizam operações aritméticas básicas: adição (+), subtração (-), multiplicação (*), divisão (/) [inteira e real], inversão de sinal (-) e resto da divisão (%).

Operadores Relacionais realizam operações de comparações lógicas entre dois operandos.

Operador	Descrição
==	Igualdade
>	Maior
<	Menor
<=	Menor igual
>=	Maior igual
!=	Diferente

Iniciação aos Algoritmos

Operadores

Operadores Aritméticos realizam operações aritméticas básicas: adição (+), subtração (-), multiplicação (*), divisão (/) [inteira e real], inversão de sinal (-) e resto da divisão (%).

Operadores Relacionais realizam operações de comparações lógicas entre dois operandos.

Operador	Descrição
==	Igualdade
>	Maior
<	Menor
<=	Menor igual
>=	Maior igual
!=	Diferente

Operadores Lógicos permitem a combinação de múltiplas operações relacionais em operações mais complexas:

- AND (&&)** assume o valor verdadeiro quando TODOS os operandos forem verdadeiros
- OR (||)** assume o valor verdadeiro quando PELO MENOS um dos operandos for verdadeiro
- NOT (!)** inverte o valor lógico do operando

Iniciação aos Algoritmos

Expressões

- Uma expressão é uma sequência de operandos conectados por operadores aritméticos e/ou relacionais e/ou lógicos a fim de permitir a definição de fórmulas matemáticas.

Iniciação aos Algoritmos

Expressões

- Uma expressão é uma sequência de operandos conectados por operadores aritméticos e/ou relacionais e/ou lógicos a fim de permitir a definição de fórmulas matemáticas.
- Um operando pode ser: um identificador (variável ou constante), um valor literal (de qualquer tipo) ou outro tipo mais complexo (a serem estudados).

Iniciação aos Algoritmos

Expressões

- Uma expressão é uma sequência de operandos conectados por operadores aritméticos e/ou relacionais e/ou lógicos a fim de permitir a definição de fórmulas matemáticas.
- Um operando pode ser: um identificador (variável ou constante), um valor literal (de qualquer tipo) ou outro tipo mais complexo (a serem estudados).

Iniciação aos Algoritmos

Expressões

- Expressões Aritméticas:

$$1 + 1$$

$$B * B - 4 * A * C$$

$$(1/2) * Base * Altura$$

Iniciação aos Algoritmos

Expressões

■ Expressões Aritméticas:

$$1 + 1$$

$$B * B - 4 * A * C$$

$$(1/2) * Base * Altura$$

■ Expressões Relacionais:

$$Media \geq 7.0$$

$$A + B \neq 0$$

Iniciação aos Algoritmos

Expressões

- Expressões Aritméticas:

$$1 + 1$$

$$B * B - 4 * A * C$$

$$(1/2) * Base * Altura$$

- Expressões Relacionais:

$$Media \geq 7.0$$

$$A + B \neq 0$$

- Expressões Lógicas:

$$(Idade > 0) \&\& (Idade \leq 21)$$

Iniciação aos Algoritmos

Operação de Atribuição

- Denomina-se de **atribuição** à operação de se 'atribuir' um valor a uma determinada variável.

Iniciação aos Algoritmos

Operação de Atribuição

- Denomina-se de **atribuição** à operação de se 'atribuir' um valor a uma determinada variável.
- É denotado através do comando

<ID variável> = <valor> ou <expressão>;

Iniciação aos Algoritmos

Operação de Atribuição

- Denomina-se de **atribuição** à operação de se 'atribuir' um valor a uma determinada variável.
- É denotado através do comando

<ID variável> = <valor> ou <expressão>;

$X = 10;$

$Media = (Nota1 + Nota2)/2.0$

Iniciação aos Algoritmos

Operação de Atribuição

- Denomina-se de **atribuição** à operação de se 'atribuir' um valor a uma determinada variável.
- É denotado através do comando

<ID variável> = <valor> ou <expressão>;

$X = 10;$

$Media = (Nota1 + Nota2)/2.0$

- **IMPORTANTE!** O tipo de <valor> precisa ser compatível com o tipo declarado da variável (isso não significa ser igual).

Iniciação aos Algoritmos

Operação de Atribuição

- Denomina-se de **atribuição** à operação de se 'atribuir' um valor a uma determinada variável.
- É denotado através do comando

<ID variável> = <valor> ou <expressão>;

$X = 10;$

$Media = (Nota1 + Nota2)/2.0$

- **IMPORTANTE!** O tipo de <valor> precisa ser compatível com o tipo declarado da variável (isso não significa ser igual).
- A definição de “compatibilidade” depende da linguagem de programação em uso. Cada uma tem as suas regras de compatibilidade específicas.

Iniciação aos Algoritmos

Operação de Atribuição

- Denomina-se de **atribuição** à operação de se 'atribuir' um valor a uma determinada variável.
- É denotado através do comando

<ID variável> = <valor> ou <expressão>;

X = 10;

Media = (Nota1 + Nota2)/2.0

- **IMPORTANTE!** O tipo de <valor> precisa ser compatível com o tipo declarado da variável (isso não significa ser igual).
- A definição de “compatibilidade” depende da linguagem de programação em uso. Cada uma tem as suas regras de compatibilidade específicas.
- Por exemplo: em linguagem C é compatível atribuir um valor inteiro a uma variável do tipo real e vice-versa (conversão dinâmica de tipos).

Iniciação aos Algoritmos

Ordem de Precedência das Operações

1. Parênteses ()
2. Operador unário (inversor de sinal): -
3. Operadores multiplicação (*), divisão (/) e resto (%)
4. Operadores adição (+) e subtração (-)
5. Operadores relacionais(==, !=, >, <, >=, <=)
6. Operadores lógicos (&&, ||, !)
7. Operador de atribuição (=)

Iniciação aos Algoritmos

Assumindo que as variáveis a , b , c são do tipo inteiro e as variáveis x , y , z são reais, qual o tipo resultante das expressões abaixo?

1. $a + b * c$
2. $a + b + y$
3. a / b
4. a / z
5. x / y
6. $a \% b + c$
7. $a + b + x > z$

Iniciação aos Algoritmos

Ordem de Precedência das Operações

Qual o resultado atribuído à variável *Resultado* em cada expressão a seguir?
Assuma: $X = 1$, $Y = 2$ e $Z = 3$

1. $Resultado = X + 5 * Y$
2. $Resultado = -(10 + Z) * 2 + X$
3. $Resultado = -(10 + Z) * 2 + X > 0$
4. $Resultado = X + Y > Z \ \&\& \ Z - X \% Y == 0$

Iniciação aos Algoritmos

Escreva as fórmulas abaixo através da notação de expressões e atribuições:

1. $x = \frac{2+a}{b-3} - 2x + x^2$

Iniciação aos Algoritmos

Escreva as fórmulas abaixo através da notação de expressões e atribuições:

1. $x = \frac{2+a}{B-3} - 2x + x^2$

$$x = ((2+a)/(B-3)) - 2*x + x*x;$$

Iniciação aos Algoritmos

Escreva as fórmulas abaixo através da notação de expressões e atribuições:

1. $x = \frac{2+a}{B-3} - 2x + x^2$

$$x = ((2+a)/(B-3)) - 2*x + x*x;$$

2. $y = \frac{\frac{2}{3x} + 4}{\frac{x}{2}}$

Iniciação aos Algoritmos

Escreva as fórmulas abaixo através da notação de expressões e atribuições:

1. $x = \frac{2+a}{B-3} - 2x + x^2$

$$x = ((2+a)/(B-3)) - 2*x + x*x;$$

2. $y = \frac{\frac{2}{3x} + 4}{\frac{x}{2}}$

$$y = (2/(3*x)+4)/(x/2);$$

Iniciação aos Algoritmos

Escreva as fórmulas abaixo através da notação de expressões e atribuições:

1. $x = \frac{2+a}{B-3} - 2x + x^2$

$$x = ((2+a)/(B-3)) - 2*x + x*x;$$

2. $y = \frac{\frac{2}{3x} + 4}{\frac{x}{2}}$

$$y = (2/(3*x)+4)/(x/2);$$

3. $z = -\frac{\frac{x^3+4y}{a+b+c}}{3z-xyz}$

Iniciação aos Algoritmos

Escreva as fórmulas abaixo através da notação de expressões e atribuições:

$$1. \ x = \frac{2+a}{B-3} - 2x + x^2$$

$$x = ((2+a)/(B-3)) - 2*x + x*x;$$

$$2. \ y = \frac{\frac{2}{3x} + 4}{\frac{x}{2}}$$

$$y = (2/(3*x)+4)/(x/2);$$

$$3. \ z = -\frac{\frac{x^3+4y}{a+b+c}}{3z-xyz}$$

$$z = -(((x*x*x + 4*y)/(a+b+c))/(3*z-x*y*z));$$

Iniciação aos Algoritmos

Indique o valor que cada uma variáveis armazenará ao final da execução deste programa

```
int A = 1, B = 2, C = 3;  
float D;
```

```
A = A + B + C;  
B = A + B + C;  
C = A + B + C;  
D = A + B + C;
```

Iniciação aos Algoritmos

Comandos de Entrada e Saída

Entrada de Dados operação onde valores fornecidos pelo usuário são armazenados pelo programa em variáveis.

Iniciação aos Algoritmos

Comandos de Entrada e Saída

Entrada de Dados operação onde valores fornecidos pelo usuário são armazenados pelo programa em variáveis.
Representa as informações necessárias para a correta execução do processamento de um programa.

Iniciação aos Algoritmos

Comandos de Entrada e Saída

Entrada de Dados operação onde valores fornecidos pelo usuário são armazenados pelo programa em variáveis.
Representa as informações necessárias para a correta execução do processamento de um programa.
Utiliza os dispositivos de entrada usuais: teclado, mouse, etc.

Iniciação aos Algoritmos

Comandos de Entrada e Saída

Entrada de Dados operação onde valores fornecidos pelo usuário são armazenados pelo programa em variáveis.

Representa as informações necessárias para a correta execução do processamento de um programa.

Utiliza os dispositivos de entrada usuais: teclado, mouse, etc.

O fluxo de informação segue sempre a seguinte sequência:

usuário → buffer do dispositivo de entrada → sistema operacional → variáveis do programa

Iniciação aos Algoritmos

Comandos de Entrada e Saída

- em C:

```
scanf("<formato>", &<variável> );
```


Iniciação aos Algoritmos

Comandos de Entrada e Saída

- em C:

scanf("<formato>", &<variável>);

- Formatos válidos:

Formato	Tipo de dado
%i	inteiro
%d	decimal (inteiro)
%f	float
%c	caracter
%s	string (cadeia de caracteres)

Iniciação aos Algoritmos

Comandos de Entrada e Saída

- em C:

```
scanf("<formato>", &<variável> );
```

- Formatos válidos:

Formato	Tipo de dado
%i	inteiro
%d	decimal (inteiro)
%f	float
%c	caracter
%s	string (cadeia de caracteres)

- Exemplos:

```
scanf("%d", &x);
scanf("%f", &nota_aluno);
```

Iniciação aos Algoritmos

Comandos de Entrada e Saída

- em C:

```
scanf("<formato>", &<variável> );
```

- Formatos válidos:

Formato	Tipo de dado
%i	inteiro
%d	decimal (inteiro)
%f	float
%c	caracter
%s	string (cadeia de caracteres)

- Exemplos:

```
scanf("%d", &x);
scanf("%f", &nota_aluno);
```

ATENÇÃO! Para se efetuar leitura de variáveis do tipo '*string*' pode-se também utilizar o comando

```
gets(<var.>);
```

Iniciação aos Algoritmos

Comandos de Entrada e Saída

Saída de Resultados representa qualquer informação fornecida como retorno ao usuário.

Iniciação aos Algoritmos

Comandos de Entrada e Saída

Saída de Resultados representa qualquer informação fornecida como retorno ao usuário.

Utiliza os dispositivos de saída: monitor, impressora, etc.

Iniciação aos Algoritmos

Comandos de Entrada e Saída

Saída de Resultados representa qualquer informação fornecida como retorno ao usuário.

Utiliza os dispositivos de saída: monitor, impressora, etc.

O fluxo de informação segue sempre a seguinte sequência:

programa → sistema operacional → buffer do dispositivo de saída → usuário

Iniciação aos Algoritmos

Comandos de Entrada e Saída

em C:

```
printf( <expressão> );
```

Iniciação aos Algoritmos

Comandos de Entrada e Saída

em C:

printf(<expressão>);

Exemplos:

```
printf("Ola Mundo da ALP!\n\n");
```


Iniciação aos Algoritmos

Comandos de Entrada e Saída

em C:

printf(<expressão>);

Exemplos:

```
printf("Ola Mundo da ALP!\n\n");  
printf("\nResultado final:  %d\n", resultado);
```

Iniciação aos Algoritmos

Comandos de Entrada e Saída

em C:

printf(<expressão>);

Exemplos:

```
printf("Ola Mundo da ALP!\n\n");  
printf("\nResultado final:  %d\n", resultado);  
printf("Nota obtida =  %.2f\n", nota);
```

Iniciação aos Algoritmos

Manipulação de Buffers: Inundação de Buffer

- Geralmente, os *buffers de memória de E/S* são manipulados através dos dispositivos específicos de E/S porém, existem diversas situações onde será desejável termos o controle sobre seus conteúdos (enchimento e esvaziamento de buffers).

Iniciação aos Algoritmos

Manipulação de Buffers: Inundação de Buffer

- Geralmente, os *buffers de memória de E/S* são manipulados através dos dispositivos específicos de E/S porém, existem diversas situações onde será desejável termos o controle sobre seus conteúdos (enchimento e esvaziamento de buffers).
- Considere a situação onde a quantidade de informação a ser fornecida como entrada é muito grande (ou onde a mesma aplicação precisa ser executada múltiplas vezes).

Iniciação aos Algoritmos

Manipulação de Buffers: Inundação de Buffer

- Geralmente, os *buffers de memória de E/S* são manipulados através dos dispositivos específicos de E/S porém, existem diversas situações onde será desejável termos o controle sobre seus conteúdos (enchimento e esvaziamento de buffers).
- Considere a situação onde a quantidade de informação a ser fornecida como entrada é muito grande (ou onde a mesma aplicação precisa ser executada múltiplas vezes).

A fim de se evitar o esforço repetitivo de digitação, uma técnica clássica para entrada de dados é a “*inundação do buffer do teclado*”.

Iniciação aos Algoritmos

Manipulação de Buffers: Inundação de Buffer

- Geralmente, os *buffers de memória de E/S* são manipulados através dos dispositivos específicos de E/S porém, existem diversas situações onde será desejável termos o controle sobre seus conteúdos (enchimento e esvaziamento de buffers).
- Considere a situação onde a quantidade de informação a ser fornecida como entrada é muito grande (ou onde a mesma aplicação precisa ser executada múltiplas vezes).

A fim de se evitar o esforço repetitivo de digitação, uma técnica clássica para entrada de dados é a "*inundação do buffer do teclado*".

- Consiste em se armazenar os dados a serem digitados em um arquivo texto, e então, ao invés de digitá-los um a um, transfere-se o conteúdo do arquivo para o buffer do teclado.

Iniciação aos Algoritmos

Manipulação de Buffers: Inundação de Buffer

- Geralmente, os *buffers de memória de E/S* são manipulados através dos dispositivos específicos de E/S porém, existem diversas situações onde será desejável termos o controle sobre seus conteúdos (enchimento e esvaziamento de buffers).
- Considere a situação onde a quantidade de informação a ser fornecida como entrada é muito grande (ou onde a mesma aplicação precisa ser executada múltiplas vezes).

A fim de se evitar o esforço repetitivo de digitação, uma técnica clássica para entrada de dados é a “*inundação do buffer do teclado*”.

- Consiste em se armazenar os dados a serem digitados em um arquivo texto, e então, ao invés de digitá-los um a um, transfere-se o conteúdo do arquivo para o buffer do teclado.

Utilizando a linha de comando (Em Windows: Prompt de Comando e em Linux: janela Terminal):

[nome do programa] < [arquivo de entrada]

Iniciação aos Algoritmos

Manipulação de Buffers: Inundação de Buffer

- Geralmente, os *buffers de memória de E/S* são manipulados através dos dispositivos específicos de E/S porém, existem diversas situações onde será desejável termos o controle sobre seus conteúdos (enchimento e esvaziamento de buffers).
- Considere a situação onde a quantidade de informação a ser fornecida como entrada é muito grande (ou onde a mesma aplicação precisa ser executada múltiplas vezes).

A fim de se evitar o esforço repetitivo de digitação, uma técnica clássica para entrada de dados é a “*inundação do buffer do teclado*”.

- Consiste em se armazenar os dados a serem digitados em um arquivo texto, e então, ao invés de digitá-los um a um, transfere-se o conteúdo do arquivo para o buffer do teclado.

Utilizando a linha de comando (Em Windows: Prompt de Comando e em Linux: janela Terminal):

[nome do programa] < [arquivo de entrada]

- Analogamente, para o buffer de saída: [nome do programa] > [arquivo de saída]

Iniciação aos Algoritmos

Manipulação de Buffers: Inundação de Buffer

- Geralmente, os *buffers de memória de E/S* são manipulados através dos dispositivos específicos de E/S porém, existem diversas situações onde será desejável termos o controle sobre seus conteúdos (enchimento e esvaziamento de buffers).
- Considere a situação onde a quantidade de informação a ser fornecida como entrada é muito grande (ou onde a mesma aplicação precisa ser executada múltiplas vezes).

A fim de se evitar o esforço repetitivo de digitação, uma técnica clássica para entrada de dados é a “*inundação do buffer do teclado*”.

- Consiste em se armazenar os dados a serem digitados em um arquivo texto, e então, ao invés de digitá-los um a um, transfere-se o conteúdo do arquivo para o buffer do teclado.

Utilizando a linha de comando (Em Windows: Prompt de Comando e em Linux: janela Terminal):

[nome do programa] < [arquivo de entrada]

- Analogamente, para o buffer de saída: [nome do programa] > [arquivo de saída]
Neste caso, todas as informações direcionadas ao buffer de saída serão armazenadas no arquivo de saída especificado.

Iniciação aos Algoritmos

Manipulação de Buffers: Esvaziamento de buffer

- O esvaziamento forçado de um buffer também é possível através do comando em C: **fflush(stdin);** para o buffer de entrada e **fflush(stdout);** para o buffer de saída.

Iniciação aos Algoritmos

Manipulação de Buffers: Esvaziamento de buffer

- O esvaziamento forçado de um buffer também é possível através do comando em C: **fflush(stdin);** para o buffer de entrada e **fflush(stdout);** para o buffer de saída.
- Esvaziar um buffer de entrada é útil para garantirmos que nenhum caractere remanescente de outra execução do programa esteja ainda armazenado no buffer, o que poderia causar erro durante a leitura de dados.

Iniciação aos Algoritmos

Manipulação de Buffers: Esvaziamento de buffer

- O esvaziamento forçado de um buffer também é possível através do comando em C: **fflush(stdin);** para o buffer de entrada e **fflush(stdout);** para o buffer de saída.
- Esvaziar um buffer de entrada é útil para garantirmos que nenhum caractere remanescente de outra execução do programa esteja ainda armazenado no buffer, o que poderia causar erro durante a leitura de dados.
- Esvaziar um buffer de saída é aconselhável quando o destino da saída é um arquivo em disco (e não a tela do monitor). O esvaziamento forçado garante que os dados sejam realmente gravados no disco pelo sistema operacional.

Iniciação aos Algoritmos

Manipulação de Buffers: Esvaziamento de buffer

- O esvaziamento forçado de um buffer também é possível através do comando em C: **fflush(stdin);** para o buffer de entrada e **fflush(stdout);** para o buffer de saída.
- Esvaziar um buffer de entrada é útil para garantirmos que nenhum caracter remanescente de outra execução do programa esteja ainda armazenado no buffer, o que poderia causar erro durante a leitura de dados.
- Esvaziar um buffer de saída é aconselhável quando o destino da saída é um arquivo em disco (e não a tela do monitor). O esvaziamento forçado garante que os dados sejam realmente gravados no disco pelo sistema operacional.
- **AVISO!** Em Linux é aconselhado **não** usar esvaziamento forçado de buffer, dado que o sistema operacional geralmente faz um bom trabalho em garantir a limpeza dos buffers. O mesmo não acontece sempre com Windows; onde com uma certa frequência é necessário o uso desses comandos para a correta liberação da memória.

Iniciação aos Algoritmos

Linguagem C

```
#include <stdio.h>

int main()
{
    printf("\n  Oi turma de ALP\n ");
    return(1);
}
```

Iniciação aos Algoritmos

Processo de Compilação

Linux Através de linha de comando

gcc -o < arquivo de saida > < arquivo fonte > .c

Exemplo: **gcc -o saida.out programa.c**

¹O diretório onde o gcc foi instalado deve estar presente na variável de ambiente PATH

Iniciação aos Algoritmos

Processo de Compilação

Linux Através de linha de comando

gcc -o < arquivo de saida > < arquivo fonte > .c

Exemplo: **gcc -o saida.out programa.c**

Windows Através de linha de comando¹

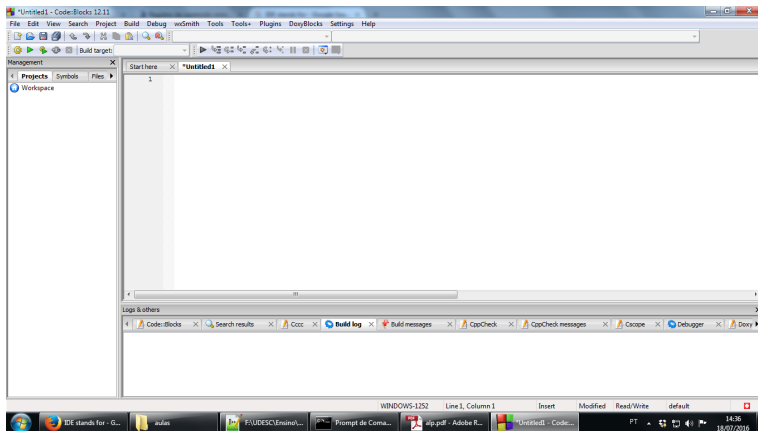
gcc -o < arquivo de saida > < arquivo fonte > .c

ou através de uma *Integrated Development Environment* (IDE) como CodeBlocks, Dev-Cpp, Eclipse, MS-Visual Studio, etc.

¹O diretório onde o gcc foi instalado deve estar presente na variável de ambiente PATH

Iniciação aos Algoritmos

Processo de Compilação



Iniciação aos Algoritmos

Interface ao Usuário

- Um programa de computador é na verdade uma sequência de instruções computacionais dividido em três partes principais:

Iniciação aos Algoritmos

Interface ao Usuário

- Um programa de computador é na verdade uma sequência de instruções computacionais dividido em três partes principais:
 1. Entrada de Dados
 2. Processamento de Dados
 3. Saída de Resultados

Iniciação aos Algoritmos

Interface ao Usuário

- Um programa de computador é na verdade uma sequência de instruções computacionais dividido em três partes principais:
 1. Entrada de Dados
 2. Processamento de Dados
 3. Saída de Resultados
- Tanto na etapa de entrada de dados quanto na de saída de resultados, o programa deve realizar uma “troca de informações” com o usuário = seja solicitar ou retornar informações.

Iniciação aos Algoritmos

Interface ao Usuário

- Um programa de computador é na verdade uma sequência de instruções computacionais dividido em três partes principais:
 1. Entrada de Dados
 2. Processamento de Dados
 3. Saída de Resultados
- Tanto na etapa de entrada de dados quanto na de saída de resultados, o programa deve realizar uma “troca de informações” com o usuário = seja solicitar ou retornar informações.
- Em ambos os casos isso é realizado através de uma **interface**. Uma interface é um descrição inteligível ao usuário sobre as ações/dados que estão sendo utilizadas pelo programa.

Iniciação aos Algoritmos

Interface ao Usuário

- Um programa de computador é na verdade uma sequência de instruções computacionais dividido em três partes principais:
 1. Entrada de Dados
 2. Processamento de Dados
 3. Saída de Resultados
- Tanto na etapa de entrada de dados quanto na de saída de resultados, o programa deve realizar uma “troca de informações” com o usuário = seja solicitar ou retornar informações.
- Em ambos os casos isso é realizado através de uma **interface**. Uma interface é um descrição inteligível ao usuário sobre as ações/dados que estão sendo utilizadas pelo programa.
- A interface pode ser *textual* (baseada em troca de mensagens de texto) ou *gráfica* (baseada em imagens gráfica).

Iniciação aos Algoritmos

Interface ao Usuário

Interface para Entrada de Dados conjunto de mensagens que informam o usuário sobre a finalidade do programa e como proceder para fornecer os dados para o início da execução do programa.

Iniciação aos Algoritmos

Interface ao Usuário

Interface para Entrada de Dados conjunto de mensagens que informam o usuário sobre a finalidade do programa e como proceder para fornecer os dados para o início da execução do programa.

Exemplo:

```
printf("Digite um número:");  
scanf("%d", &X);
```


Iniciação aos Algoritmos

Interface ao Usuário

Interface para Entrada de Dados conjunto de mensagens que informam o usuário sobre a finalidade do programa e como proceder para fornecer os dados para o início da execução do programa.

Exemplo:

```
printf("Digite um número:");  
scanf("%d", &X);
```

Interface para a saída de Resultados conjunto de mensagens que explicam o processamento que foi realizado e quais foram os resultados obtidos a partir deste (em função dos dados fornecidos na entrada de dados).

Iniciação aos Algoritmos

Interface ao Usuário

Interface para Entrada de Dados conjunto de mensagens que informam o usuário sobre a finalidade do programa e como proceder para fornecer os dados para o início da execução do programa.

Exemplo:

```
printf("Digite um número:");
scanf("%d", &X);
```

Interface para a saída de Resultados conjunto de mensagens que explicam o processamento que foi realizado e quais foram os resultados obtidos a partir deste (em função dos dados fornecidos na entrada de dados).

Exemplo:

```
printf("A média entre os números ");
printf(A);
printf(" e ");
printf(B);
printf(" é igual a ");
printf(media);
```

Iniciação aos Algoritmos

Interface ao Usuário: *Placeholders*

- O comando `printf` permite a inclusão de “*placeholders*” a fim de simplificar a edição de mensagens compostas por múltiplos tipos de dados simultaneamente.

Iniciação aos Algoritmos

Interface ao Usuário: *Placeholders*

- O comando `printf` permite a inclusão de “*placeholders*” a fim de simplificar a edição de mensagens compostas por múltiplos tipos de dados simultaneamente.
- Um “*placeholder*” é apenas uma reserva de espaço que serve como indicador de que naquele local um valor de um determinado tipo de dados será impresso. *Placeholders* são representados de maneira muito parecida ao formato que se utiliza no comando de entrada *scanf*.

Iniciação aos Algoritmos

Interface ao Usuário: *Placeholders*

- O comando `printf` permite a inclusão de “*placeholders*” a fim de simplificar a edição de mensagens compostas por múltiplos tipos de dados simultaneamente.
- Um “*placeholder*” é apenas uma reserva de espaço que serve como indicador de que naquele local um valor de um determinado tipo de dados será impresso. *Placeholders* são representados de maneira muito parecida ao formato que se utiliza no comando de entrada *scanf*.
- Exemplo:
 - `printf("Valor obtido é igual a %d", X);`

Iniciação aos Algoritmos

Interface ao Usuário: *Placeholders*

- O comando `printf` permite a inclusão de “*placeholders*” a fim de simplificar a edição de mensagens compostas por múltiplos tipos de dados simultaneamente.
- Um “*placeholder*” é apenas uma reserva de espaço que serve como indicador de que naquele local um valor de um determinado tipo de dados será impresso. *Placeholders* são representados de maneira muito parecida ao formato que se utiliza no comando de entrada *scanf*.
- Exemplo:
 - `printf("Valor obtido é igual a %d", X);`
 - `printf("A soma dos %d primeiros elementos do conjunto é %f\n", N, soma);`

Iniciação aos Algoritmos

Interface ao Usuário: *Placeholders*

- O comando `printf` permite a inclusão de “*placeholders*” a fim de simplificar a edição de mensagens compostas por múltiplos tipos de dados simultaneamente.
- Um “*placeholder*” é apenas uma reserva de espaço que serve como indicador de que naquele local um valor de um determinado tipo de dados será impresso. *Placeholders* são representados de maneira muito parecida ao formato que se utiliza no comando de entrada *scanf*.
- Exemplo:
 - `printf("Valor obtido é igual a %d", X);`
 - `printf("A soma dos %d primeiros elementos do conjunto é %f\n", N, soma);`
 - `printf("\tMédia do aluno %s\n\tNota obtida: %2.2f\n\n", Nome, media);`

Iniciação aos Algoritmos

```
#include <stdio.h>

int main()
{
    int num_inteiro;
    char sexo;
    float num_real;

    printf("\n ENTRADA d c f: ");

    scanf("%d %c %f", &num_inteiro, &sexo, &num_real);
    printf(" SAIDA: \n");
    printf(" Int: %d  Char: %c Real: %f",
        num_inteiro, sexo, num_real );

    return(1);
}
```


Iniciação aos Algoritmos

Exercícios

- Construa um programa para calcular a média aritmética entre três notas.

$$media = \frac{A + B + C}{3}$$

Iniciação aos Algoritmos

Exercícios

- Construa um programa para calcular a média aritmética entre três notas.

$$media = \frac{A + B + C}{3}$$

```
#include <stdio.h>

int main()
{
    float Nota_1, Nota_2, Media;

    printf("Objetivo: Este programa calcula a media entre duas notas:\n\n");
    printf("Digite a primeira nota: ");
    scanf("%f", &Nota_1);

    printf("Digite a segunda nota: ");
    scanf("%f", &Nota_2);

    Media = (Nota_1 + Nota_2) / 2.0;

    printf("\n\nA media entre as notas digitadas = %.1f\n", Media);

    return(1);
}
```

Iniciação aos Algoritmos

Exercícios

- Construa um algoritmo para calcular a média ponderada entre três notas.

$$media = \frac{AP_1 + BP_2 + CP_3}{P_1 + P_2 + P_3}$$

Iniciação aos Algoritmos

Exercícios

```
#include <stdio.h>

int main()
{
    float Nota_1, Nota_2, Nota_3, Media;
    int Peso_1, Peso_2, Peso_3;

    printf("Objetivo: Este programa calcula a media ponderada entre tres notas:\n\n");
    printf("Digite a primeira nota: ");
    scanf("%f", &Nota_1);

    printf("Digite o peso da primeira nota: ");
    scanf("%d", &Peso_1);

    printf("Digite a segunda nota: ");
    scanf("%f", &Nota_2);

    printf("Digite o peso da segunda nota: ");
    scanf("%d", &Peso_2);

    printf("Digite a terceira nota: ");
    scanf("%f", &Nota_3);

    printf("Digite o peso da terceira nota: ");
    scanf("%d", &Peso_3);

    Media = (Nota_1*Peso_1 + Nota_2*Peso_2 + Nota_3*Peso_3) / (Peso_1 + Peso_2 + Peso_3);

    printf("\n\nA media entre as notas digitadas = %.1f\n", Media);

    return(1);
}
```

Iniciação aos Algoritmos

Exercícios

- Considere que para um determinado funcionário se saiba:

- ☐ seu nome
- ☐ o código de sua categoria funcional (A, B ou C)
- ☐ seu salário base

Determine quanto o funcionário vai receber, dado que:

- ☐ o salário base do funcionário foi reajustado em 13%
- ☐ O funcionário recebe uma gratificação de 20% sobre o salário base
- ☐ são feitos descontos de 15% sobre o salário total a título de INSS

Iniciação aos Algoritmos

Exercícios

```
#include <stdio.h>

int main()
{
    char nome[10];
    char categoria;
    float SalarioBase, SalarioBruto, SalarioReajustado, SalarioLiquido;

    printf("Objetivo: Este programa calcula o salario de um funcionario de acordo com regras trabalhistas\n\n");
    printf("Digite o nome do funcionario: ");
    gets(nome);

    printf("Digite a categoria do funcionario (A, B ou C): ");
    scanf("%c",&categoria);

    printf("Digite o salario base do funcionario: ");
    scanf("%f",&SalarioBase);

    SalarioReajustado = SalarioBase * 1.13;
    SalarioBruto = SalarioReajustado + SalarioBase * 0.2;
    SalarioLiquido = SalarioBruto * 0.85;

    printf("\n\nNome do Funcionario: %s\n", nome);
    printf("Categoria Funcional: %c\n", categoria);
    printf("Salario Bruto: R$%.2f\n", SalarioBruto);
    printf("Salario Liquido R$%.2f\n", SalarioLiquido);

    return(1);
}
```

Iniciação aos Algoritmos

Exercícios

```
#include <stdio.h>

int main() {
    int N, M, Litros;

    printf("Objetivo: Este programa calcular a qtde de água restante num recipiente após  
o enchimento de M garrafas\n");
    printf("Digite o número de litros no recipiente:\n");
    scanf("%d", &N);

    printf("Digite a quantidade de garrafas de 1 litro:\n");
    scanf("%d", &M);

    Litros = N % M;

    printf("A quantidade de litros que restará é %d", Litros);
    return 1;
}
```

Iniciação aos Algoritmos

Exercícios

1. Convertendo temperaturas:

1.1 Faça um programa que converta temperaturas em graus Celsius (C) para Farenheit (F)

$$F = C \times \frac{9}{5} + 32$$

1.2 Faça um programa que converta temperaturas em graus Farenheit (F) para Celsius (C)

$$C = (F - 32) \times \frac{5}{9}$$

1.3 Faça um programa que converta temperaturas em graus Celsius (C) para Kelvin (K)

$$K = C + 273.15$$

1.4 Faça um programa que converta temperaturas em graus Kelvin (K) para Celsius (C)

$$C = K - 273.15$$

1.5 Faça um programa que converta temperaturas em graus Farenheit (F) para Kelvin (K)

$$K = (F + 459.67) \times \frac{5}{9}$$

1.6 Faça um programa que converta temperaturas em graus Kelvin (K) para Farenheit (F)

$$F = K \times \frac{9}{5} - 459.67$$

2. Dado que $U = R \times I$, onde U = tensão elétrica, R = resistência em ohms (Ω) e I = intensidade da corrente elétrica. Faça três programas que calcule cada um desses elementos respectivamente (dados os outros dois).

3. Faça um programa que converta velocidade dada em Km/h para Mph e para m/s.

Iniciação aos Algoritmos

Comandos de Desvio de Fluxo de Execução

- No paradigma de *programação estruturada* é denominado fluxo de execução à ordem de execução das instruções de um algoritmo.

Iniciação aos Algoritmos

Comandos de Desvio de Fluxo de Execução

- No paradigma de *programação estruturada* é denominado fluxo de execução à ordem de execução das instruções de um algoritmo.
- Tradicionalmente, temos uma execução sequencial: instrução #1 depois a instrução #2, #3, e assim por diante até o fim do algoritmo.

Iniciação aos Algoritmos

Comandos de Desvio de Fluxo de Execução

- No paradigma de *programação estruturada* é denominado fluxo de execução à ordem de execução das instruções de um algoritmo.
- Tradicionalmente, temos uma execução sequencial: instrução #1 depois a instrução #2, #3, e assim por diante até o fim do algoritmo.
- É denominado 'desvio' do fluxo de execução a qualquer alteração dessa ordem natural de execução.

Iniciação aos Algoritmos

Comandos de Desvio de Fluxo de Execução

- No paradigma de *programação estruturada* é denominado fluxo de execução à ordem de execução das instruções de um algoritmo.
- Tradicionalmente, temos uma execução sequencial: instrução #1 depois a instrução #2, #3, e assim por diante até o fim do algoritmo.
- É denominado 'desvio' do fluxo de execução a qualquer alteração dessa ordem natural de execução.
- Os comandos de desvio de fluxo são divididos em:
 - Condicional e Seleção permitem decidir qual instrução(ões) serão executadas em função de uma análise condicional.
 - Repetições permitem a execução de uma instrução(ões) múltiplas vezes.
 - Sub-Rotinas permite o reaproveitamento de trechos do código-fonte em múltiplas partes do programa sem a necessidade de reprogramação.

Iniciação aos Algoritmos

Comandos Condicionais

- Permite uma tomada de decisão e escolha de um *bloco de comandos* a ser executado condicionalmente à análise de uma expressão.

Iniciação aos Algoritmos

Comandos Condicionais

- Permite uma tomada de decisão e escolha de um *bloco de comandos* a ser executado condicionalmente à análise de uma expressão.

- Dois casos:

1. Execução condicional de um bloco de comandos

```
if(<condição>)
    <bloco de comandos>
```

Iniciação aos Algoritmos

Comandos Condicionais

- Permite uma tomada de decisão e escolha de um *bloco de comandos* a ser executado condicionalmente à análise de uma expressão.

- Dois casos:

1. Execução condicional de um bloco de comandos

```
if(<condição>)
    <bloco de comandos>
```

2. Escolha condicional entre dois blocos de comandos

```
if(<condição>)
    <bloco de comandos 1>
else
    <bloco de comandos 2>
```

Iniciação aos Algoritmos

Comandos Condicionais

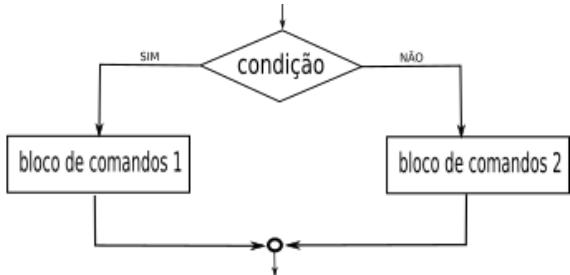
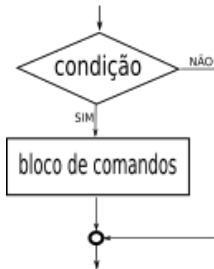
- Permite uma tomada de decisão e escolha de um *bloco de comandos* a ser executado condicionalmente à análise de uma expressão.
- Dois casos:
 1. Execução condicional de um bloco de comandos


```
if(<condição>)
    <bloco de comandos>
```
 2. Escolha condicional entre dois blocos de comandos


```
if(<condição>)
    <bloco de comandos 1>
else
    <bloco de comandos 2>
```
- **ALERTA!** Caso um bloco de comandos contenha dois ou mais comandos então **obrigatoriamente** este deve ser delimitado por { e }

Iniciação aos Algoritmos

Comandos Condicionais



Iniciação aos Algoritmos

Comandos Condicionais

```
if(idade > 65)
    printf("Pessoa Idosa");
```

Iniciação aos Algoritmos

Comandos Condicionais

```
if(idade > 65)
    printf("Pessoa Idosa");

if(media>=5.0)
{
    printf("Media: %.3f\n", media);
    printf("Aluno APROVADO");
}
else
{
    printf("Media: %.3f\n", media);
    printf("Aluno REPROVADO");
}
```

Iniciação aos Algoritmos

Problema da Classificação de Dados

- O problema da *classificação* ou problema da *múltipla escolha* consiste em determinar para qual classe um determinado atributo pertence. Em muitos casos as classes são mutuamente exclusivas (o atributo só pode pertencer a uma das classes).

Iniciação aos Algoritmos

Problema da Classificação de Dados

- O problema da *classificação* ou problema da *múltipla escolha* consiste em determinar para qual classe um determinado atributo pertence. Em muitos casos as classes são mutuamente exclusivas (o atributo só pode pertencer a uma das classes).
- Exemplos de classificação:
 - Idade de uma pessoa (criança, adolescente, jovem, adulto ou idoso);
 - Temperaturas: frio, morno ou quente;
 - Resultado semestral: reprovado, em exame ou aprovado.

Iniciação aos Algoritmos

Problema da Classificação de Dados

- O problema da *classificação* ou problema da *múltipla escolha* consiste em determinar para qual classe um determinado atributo pertence. Em muitos casos as classes são mutuamente exclusivas (o atributo só pode pertencer a uma das classes).
- Exemplos de classificação:
 - Idade de uma pessoa (criança, adolescente, jovem, adulto ou idoso);
 - Temperaturas: frio, morno ou quente;
 - Resultado semestral: reprovado, em exame ou aprovado.
- **PROBLEMA!** O comando condicional apenas permite a escolha entre DUAS opções. Como fazer para escolher entre múltiplas?

Iniciação aos Algoritmos

Problema da Classificação de Dados

- O problema da *classificação* ou problema da *múltipla escolha* consiste em determinar para qual classe um determinado atributo pertence. Em muitos casos as classes são mutuamente exclusivas (o atributo só pode pertencer a uma das classes).
- Exemplos de classificação:
 - Idade de uma pessoa (criança, adolescente, jovem, adulto ou idoso);
 - Temperaturas: frio, morno ou quente;
 - Resultado semestral: reprovado, em exame ou aprovado.
- **PROBLEMA!** O comando condicional apenas permite a escolha entre DUAS opções. Como fazer para escolher entre múltiplas?
- **RESPOSTA:** Comandos Condicionais aninhados

Iniciação aos Algoritmos

Comandos Condicionais Aninhados

- Consiste na utilização de mais de um condicional dependente do resultado da análise de outro(s)

Iniciação aos Algoritmos

Comandos Condicionais Aninhados

- Consiste na utilização de mais de um condicional dependente do resultado da análise de outro(s)

```

if(<condição-1>)
  if(<condição-2>)
    if(<condição-3>)
      <Comando-1>
    else
      <Comando-2>
  else
    if(<condição-4>)
      <Comando-4>
    else
      <Comando-5>
else
  <Comando-6>

```

Iniciação aos Algoritmos

Exercícios

- Adapte o programa do cálculo da média aritmética para que também informe a situação final do aluno dado que:

APROVADO se $media \geq 7.0$

EM EXAME se $2.0 \leq media < 7.0$

REPROVADO se $media < 2.0$

Iniciação aos Algoritmos

Exercícios

```
... // ver exercicio anterior sobre media aritmetica

printf("\n\nA media entre as notas digitadas = %.1f\n", Media);

if(Media >= 7)
    printf("\n0 aluno esta APROVADO");
else
    if(Media >= 2)
        printf("\n0 aluno esta EM EXAME");
    else
        printf("\n0 aluno esta REPROVADO");

return(1);
}
```

Iniciação aos Algoritmos

Exercícios

1. Faça um programa que, dados os três lados de um triângulo (a, b, c), decida se os comprimentos realmente forma um triângulo conforme a regra abaixo:

$$|b - c| < a < b + c \quad |a - c| < b < a + c \quad |a - b| < c < a + b$$

2. Adapte o programa anterior para, caso os valores fornecidos formarem um triângulo, decida se o mesmo é:

Equilátero possui os três lados iguais

Isósceles possui dois lados iguais

Escaleno possui os três lados distintos

3. Faça um programa que solicite ao usuário a escolha de uma forma geométrica: triângulo, quadrilátero ou circunferência. Em seguida, de acordo com a escolha determine a área da forma geométrica escolhida. Dados que:

Triângulo $Area = \frac{Base \times Altura}{2}$

Quadrilátero $Area = Base \times Altura$

Circunferência $Area = \pi \times Raio^2$

Iniciação aos Algoritmos

Comando de Seleção

- Uma outra forma de se resolver o problema da classificação que funciona para os casos onde o atributo a ser classificado é de um tipo enumerável (inteiro ou caracter), é através do comando de **seleção**.

Iniciação aos Algoritmos

Comando de Seleção

- Uma outra forma de se resolver o problema da classificação que funciona para os casos onde o atributo a ser classificado é de um tipo enumerável (inteiro ou caracter), é através do comando de **seleção**.
- O comando de seleção consiste em uma forma compacta de realizar múltiplas escolhas sobre uma mesma variável.

Iniciação aos Algoritmos

Comando de Seleção

- Uma outra forma de se resolver o problema da classificação que funciona para os casos onde o atributo a ser classificado é de um tipo enumerável (inteiro ou caracter), é através do comando de **seleção**.
- O comando de seleção consiste em uma forma compacta de realizar múltiplas escolhas sobre uma mesma variável.

```
switch(<Expressão>) {
case <constante 1>:
    <bloco de comandos 1>
break;
case <constante 2>:
    <bloco de comandos 2>
break;
case <constante 3>:
    <bloco de comandos 3>
break;
default:
    <bloco de comandos 4>
}
```

Iniciação aos Algoritmos

Exemplo de Comando de Seleção em C

```
switch(option) {  
  case 1:  
    printf("A opção 1 foi escolhida");  
    break;  
  case 2:  
    printf("A opção 2 foi escolhida");  
    break;  
  case 3:  
    printf("A opção 3 foi escolhida");  
    break;  
  default:  
    printf("Opção invalida");  
}
```

NOTA: Analogamente ao comando condicional, a cláusula 'default' é opcional.

Iniciação aos Algoritmos

Exercícios

1. Adapte o último programa sobre formas geométricas: triângulo, quadrilátero ou circunferência; para que utilize o comando de seleção.

Iniciação aos Algoritmos

Exercícios

1. Adapte o último programa sobre formas geométricas: triângulo, quadrilátero ou circunferência; para que utilize o comando de seleção.

```
#include <stdio.h>

int main() {
    int opcao;

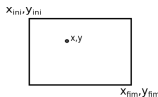
    printf("OBJETIVO: Calcular a area de formas geometricas\n");
    printf("MENU:\n\t[1] Triangulo\n\t[2] Quadrilatero\n\t[3] Circunferencia\n\n");
    printf("Opção: ");
    scanf("%d", &opcao);

    switch(opcao) {
        case 1:
            // DIGITE AQUI SEU PROCESSAMENTO SOBRE TRIANGULOS
            break;
        case 2:
            // DIGITE AQUI SEU PROCESSAMENTO SOBRE QUADRILATEROS
            break;
        case 3:
            // DIGITE AQUI SEU PROCESSAMENTO SOBRE CIRCUNFERENCIAS
            break;
        default:
            printf("Opção Invalida!");
            break;
    }
    return 1;
}
```

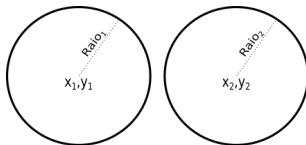
Iniciação aos Algoritmos

Exercícios

1. Dadas as coordenadas de um retângulo (x_{ini} , y_{ini} , x_{fim} , y_{fim}) e as coordenadas de um ponto (x , y). Faça um algoritmo que determine se o ponto é interno ou externo ao retângulo.



2. Dadas as coordenadas (x_1 , y_1 , $Raio_1$) e (x_2 , y_2 , $Raio_2$) que descrevem duas circunferências. Faça um algoritmo que decida se as mesmas se interceptam ou não.



3. Crie um algoritmo que permita o cálculo de conversão de temperaturas de um padrão qualquer para outro em função de uma escolha prévia feita pelo usuário (menu de opções):

$$\begin{array}{ll} C \rightarrow F & F \rightarrow C \\ C \rightarrow K & K \rightarrow C \\ F \rightarrow K & K \rightarrow F \end{array}$$

Iniciação aos Algoritmos

Exercícios

1. Considere o algoritmo abaixo. Supondo que sejam fornecidos como valores de entrada os números (...) e (...) nessa ordem, qual será a informação produzida na saída?

```
#include <stdio.h>

int main() {
    int a, b, c, d;
    scanf("%d %d", &a, &b);

    c = 2*a+4*b;
    d = a*b+c/2;
    b = a+b;
    a = 50;

    printf("%d %d %d %d", a, b, c, d);

    if(a<b && c!=d)
        if(!(a==b || a==c))
            printf("Saida #1");
        else
            printf("Saida #2");
    else
        if(a>b-d && a+b<=c*d)
            printf("Saida #3");
        else
            printf("Saida #4");

    return 1;
}
```

Iniciação aos Algoritmos

Exercício

1. Faça um algoritmo que calcule a média aritmética entre duas notas e apresente o resultado. O programa deve permitir que sejam fornecidas as notas para 3 alunos distintos.

Iniciação aos Algoritmos

Exercício

1. Faça um algoritmo que calcule a média aritmética entre duas notas e apresente o resultado. O programa deve permitir que sejam fornecidas as notas para 3 alunos distintos.

```
#include <stdio.h>

int main() {
    float nota1_aluno1, nota2_aluno1, nota1_aluno2, nota2_aluno2, nota1_aluno3, nota2_aluno3;
    float media_aluno1, media_aluno2, media_aluno3;

    printf("Digite as notas do aluno 1:");
    scanf("%f %f", &nota1_aluno1, &nota2_aluno1);
    media_aluno1 = (nota1_aluno1 + nota2_aluno1)/2;
    printf("Media do aluno 1 = %f", media_aluno1);

    printf("Digite as notas do aluno 2:");
    scanf("%f %f", &nota1_aluno2, &nota2_aluno2);
    media_aluno2 = (nota1_aluno2 + nota2_aluno2)/2;
    printf("Media do aluno 2 = %f", media_aluno2);

    printf("Digite as notas do aluno 3:");
    scanf("%f %f", &nota1_aluno3, &nota2_aluno3);
    media_aluno3 = (nota1_aluno3 + nota2_aluno3)/2;
    printf("Media do aluno 3 = %f", media_aluno3);

    return 1;
}
```

Iniciação aos Algoritmos

Comandos de Desvio de Fluxo de Execução (2)

- Os comandos de desvio de fluxo são divididos em:

Condicional e Seleção permitem decidir qual instrução(ões) serão executadas em função de uma análise condicional.

Repetições **permitem a execução de uma instrução(ões) múltiplas vezes.**

Sub-Rotinas permite o reaproveitamento de trechos do código-fonte em múltiplas partes do programa sem a necessidade de reprogramação.

Iniciação aos Algoritmos

Comandos de Repetição

- Comandos de repetição são comandos de desvio de fluxo que permitem que um mesmo comando (ou bloco de comandos) possa ser executado mais de uma vez em sequência, evitando-se assim a necessidade de reprogramação.

Iniciação aos Algoritmos

Comandos de Repetição

- Comandos de repetição são comandos de desvio de fluxo que permitem que um mesmo comando (ou bloco de comandos) possa ser executado mais de uma vez em sequência, evitando-se assim a necessidade de reprogramação.
- São também chamados de *laços de repetição*.

Iniciação aos Algoritmos

Comandos de Repetição

- Comandos de repetição são comandos de desvio de fluxo que permitem que um mesmo comando (ou bloco de comandos) possa ser executado mais de uma vez em sequência, evitando-se assim a necessidade de reprogramação.
- São também chamados de *laços de repetição*.
- Os comandos de repetição são divididos em:
 - Repetição com pré-teste** primeiro avaliam uma expressão antes de decidir se o comando/bloco será executado
 - Repetição com pós-teste** primeiro executam um comando/bloco para só então avaliar uma expressão a fim de decidir se o repete ou não
 - Repetições contadas** executam um comando/bloco um número pré-determinado de vezes.

Iniciação aos Algoritmos

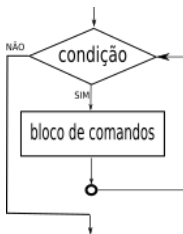
Comandos de Repetição com Pré-Teste

- A repetição com **pré-teste** ou repetição **Enquanto - faça** é aquela que avalia o resultado de uma expressão a fim de decidir se um bloco de comandos será o ou não executado.

Iniciação aos Algoritmos

Comandos de Repetição com Pré-Teste

- A repetição com **pré-teste** ou repetição **Enquanto - faça** é aquela que avalia o resultado de uma expressão a fim de decidir se um bloco de comandos será o ou não executado.
- Após a execução do bloco, o programa retorna ao ponto de avaliar novamente a condição para decidir acerca de um novo *loop* (laço de execução)



Iniciação aos Algoritmos

Comandos de Repetição com Pré-Teste

```
x = 0;
while(x < 10) {
    printf("X = %d\n", x);
    x ++; // comando equivalente em C a x = x + 1;
}
```

Iniciação aos Algoritmos

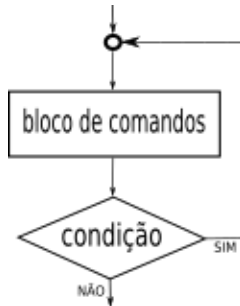
Comandos de Repetição com Pós-Teste

- A repetição com **pós-teste** ou repetição **Faça - Enquanto** é aquela executa um bloco de comandos e então avalia o resultado de uma expressão a fim de decidir se um bloco de comandos será ou não executado.

Iniciação aos Algoritmos

Comandos de Repetição com Pós-Teste

- A repetição com **pós-teste** ou repetição **Faça - Enquanto** é aquela executa um bloco de comandos e então avalia o resultado de uma expressão a fim de decidir se um bloco de comandos será ou não executado.



Iniciação aos Algoritmos

Comandos de Repetição com Pós-Teste

```
x = 0;
do {
    printf("X = %d\n", x);
    x ++; // comando equivalente em C a x = x + 1;
} while(x < 10);
```


Iniciação aos Algoritmos

Comandos de Repetição Contada

- Em determinadas situações, o número de vezes que se deseja repetir o bloco de comandos é conhecido. Nestes casos, uma alternativa é utilizar a repetição contada que, como o nome sugere, repete um bloco de comandos uma quantidade pré-determinada de vezes.

Iniciação aos Algoritmos

Comandos de Repetição Contada

- Em determinadas situações, o número de vezes que se deseja repetir o bloco de comandos é conhecido. Nestes casos, uma alternativa é utilizar a repetição contada que, como o nome sugere, repete um bloco de comandos uma quantidade pré-determinada de vezes.
- Este tipo de repetição é também conhecida como **Para - até - faça**.

Iniciação aos Algoritmos

Comandos de Repetição Contada

- Em determinadas situações, o número de vezes que se deseja repetir o bloco de comandos é conhecido. Nestes casos, uma alternativa é utilizar a repetição contada que, como o nome sugere, repete um bloco de comandos uma quantidade pré-determinada de vezes.
- Este tipo de repetição é também conhecida como **Para - até - faça**.

```
for(x = 0; x < 10; x++)  
    printf("X = %d\n", x);
```

Iniciação aos Algoritmos

Exercícios

1. Refaça o algoritmo anterior da média aritmética para 3 alunos, porém agora utilizando comandos de repetição.

Iniciação aos Algoritmos

Exercícios

1. Refaça o algoritmo anterior da média aritmética para 3 alunos, porém agora utilizando comandos de repetição.

```
#include <stdio.h>

int main() {
    float nota1, nota2, media;
    int contador;

    printf("OBJETIVO: Calcular a media aritmetica entre 2 notas para 3 alunos distintos\n\n");

    for(contador = 1; contador <= 3; contador++) {
        printf("Digite as notas do aluno %d\n", contador);
        scanf("%f %f", &nota1, &nota2);
        media = (nota1 + nota2) / 2;
        printf("Media do aluno %d = %.2f\n\n", contador, media);
    }
    return 1;
}
```

Iniciação aos Algoritmos

Exercícios

1. Adapte o algoritmo anterior para que o usuário possa informar quantos alunos existem na sala de aula.

Iniciação aos Algoritmos

Exercícios

1. Adapte o algoritmo anterior para que o usuário possa informar quantos alunos existem na sala de aula.
2. Faça um algoritmo que apresente a soma dos N primeiros números inteiros.

Iniciação aos Algoritmos

Exercícios

1. Adapte o algoritmo anterior para que o usuário possa informar quantos alunos existem na sala de aula.
2. Faça um algoritmo que apresente a soma dos N primeiros números inteiros.
3. Adapte o exercício 1 para que exiba (ao final do processo) a média geral da turma inteira.

Iniciação aos Algoritmos

Exercícios

1. Adapte o algoritmo anterior para que o usuário possa informar quantos alunos existem na sala de aula.
2. Faça um algoritmo que apresente a soma dos N primeiros números inteiros.
3. Adapte o exercício 1 para que exiba (ao final do processo) a média geral da turma inteira.
4. Faça um algoritmo que calcule o fatorial de um número N . Exemplo:

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

Unidade 04

Estruturas homogêneas e Sub-rotinas

Previsão: 26 horas/aula

Estruturas homogêneas

- Em programação o armazenamento e manipulação de dados e informações é feito através das estruturas de armazenamento.

Estruturas homogêneas

- Em programação o armazenamento e manipulação de dados e informações é feito através das estruturas de armazenamento.
- Até o momento a única estrutura vista são as variáveis (e constantes).

Estruturas homogêneas

- Em programação o armazenamento e manipulação de dados e informações é feito através das estruturas de armazenamento.
- Até o momento a única estrutura vista são as variáveis (e constantes).
- Estas estruturas permitem armazenar **UMA** informação de cada vez (mono-valor) de **UM** único tipo de dados (homogêneas).

Estruturas homogêneas

- Em programação o armazenamento e manipulação de dados e informações é feito através das estruturas de armazenamento.
- Até o momento a única estrutura vista são as variáveis (e constantes).
- Estas estruturas permitem armazenar **UMA** informação de cada vez (mono-valor) de **UM** único tipo de dados (homogêneas).
- Porém, em programação existem 4 tipos diferentes de estruturas de armazenamento de dados.

Estruturas homogêneas

Homogêneas trabalham com um único tipo de dados por vez.

Heterogêneas permitem a manipulação de múltiplos tipos de dados simultaneamente.

	Estruturas	
	Homogêneas	Heterogêneas
Mono-valor	variável	registro ponteiros
Multi-valor	vetor	Tipos Abstratos de Dados (TAD)

Estruturas homogêneas

Variáveis

- Uma variável (ou constante) é um tipo mono-valor homogêneo de dados.

Estruturas homogêneas

Variáveis

- Uma variável (ou constante) é um tipo mono-valor homogêneo de dados.
- Significa que pode armazenar um único valor por vez, de um único tipo de dados.

Estruturas homogêneas

Variáveis

- Uma variável (ou constante) é um tipo mono-valor homogêneo de dados.
- Significa que pode armazenar um único valor por vez, de um único tipo de dados.
- Declaração de uma variável:

`<TIPO> <Identificador> = <VALOR>;`

Estruturas homogêneas

Variáveis

- Uma variável (ou constante) é um tipo mono-valor homogêneo de dados.
- Significa que pode armazenar um único valor por vez, de um único tipo de dados.
- Declaração de uma variável:

`<TIPO> <Identificador> = <VALOR>;`

- Exemplo de declaração: `int x; float y = 0.04; char c = 'E';`

Estruturas homogêneas

Variáveis

- Uma variável (ou constante) é um tipo mono-valor homogêneo de dados.
- Significa que pode armazenar um único valor por vez, de um único tipo de dados.
- Declaração de uma variável:

`<TIPO> <Identificador> = <VALOR>;`

- Exemplo de declaração: `int x; float y = 0.04; char c = 'E';`
- Exemplo de uso de uma variável: `x = 5; y = A + B*2;`

Estruturas homogêneas

Variáveis

$$x = 0;$$

Estruturas homogêneas

Variáveis

`x = 0;`

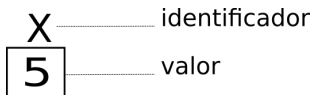
`x = 5;`

Estruturas homogêneas

Variáveis

x = 0;

x = 5;



Estruturas homogêneas

Vetores

- As estruturas homogêneas multivaloradas permitem o armazenamento de múltiplos valores simultaneamente. **Todos do mesmo tipo.**

Estruturas homogêneas

Vetores

- As estruturas homogêneas multivaloradas permitem o armazenamento de múltiplos valores simultaneamente. **Todos do mesmo tipo.**
- São denominadas **vetores** ou *arrays*.

Estruturas homogêneas

Vetores

- As estruturas homogêneas multivaloradas permitem o armazenamento de múltiplos valores simultaneamente. **Todos do mesmo tipo.**
- São denominadas **vetores** ou *arrays*.
- Declaração de um vetor:

<TIPO> <Identificador> [<tamanho>] ;

Estruturas homogêneas

Vetores

- As estruturas homogêneas multivaloradas permitem o armazenamento de múltiplos valores simultaneamente. **Todos do mesmo tipo.**
- São denominadas **vetores** ou *arrays*.
- Declaração de um vetor:

<TIPO> <Identificador> [<tamanho>] ;

- Exemplo de declaração de um vetor: `int vet[8];`

Estruturas homogêneas

Vetores

- As estruturas homogêneas multivaloradas permitem o armazenamento de múltiplos valores simultaneamente. **Todos do mesmo tipo.**
- São denominadas **vetores** ou *arrays*.
- Declaração de um vetor:

`<TIPO> <Identificador> [<tamanho>] ;`

- Exemplo de declaração de um vetor: `int vet[8];`
- Exemplo de uso de um vetor: `vet[0] = 1; vet[6] = 3;`

Estruturas homogêneas

Vetores

- As estruturas homogêneas multivaloradas permitem o armazenamento de múltiplos valores simultaneamente. **Todos do mesmo tipo.**
- São denominadas **vetores** ou *arrays*.
- Declaração de um vetor:

`<TIPO> <Identificador> [<tamanho>] ;`

- Exemplo de declaração de um vetor: `int vet[8];`
- Exemplo de uso de um vetor: `vet[0] = 1; vet[6] = 3;`
- Cada índice do vetor funciona como uma variável (pode armazenar um valor de cada vez).

Estruturas homogêneas

Vetores

- As estruturas homogêneas multivaloradas permitem o armazenamento de múltiplos valores simultaneamente. **Todos do mesmo tipo.**
- São denominadas **vetores** ou *arrays*.
- Declaração de um vetor:

`<TIPO> <Identificador> [<tamanho>] ;`

- Exemplo de declaração de um vetor: `int vet[8];`
- Exemplo de uso de um vetor: `vet[0] = 1; vet[6] = 3;`
- Cada índice do vetor funciona como uma variável (pode armazenar um valor de cada vez).
- O primeiro índice do vetor é sempre indicado pelo valor zero: `vet[0]`

Estruturas homogêneas

Vetores

```
vet[0] = 1;
```

Estruturas homogêneas

Vetores

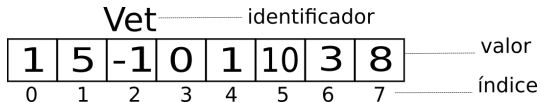
```
vet[0] = 1;  
vet[6] = 3;
```


Estruturas homogêneas

Vetores

`vet[0] = 1;`

`vet[6] = 3;`



Estruturas homogêneas

Vetores

```
#include <stdio.h>

#define TAM 10

int main(){
    int vet[TAM], i;

    printf("OBJETIVO: Este programa mostra um vetor de %d inteiros invertido\n", TAM);
    printf("Digite o vetor:\n");

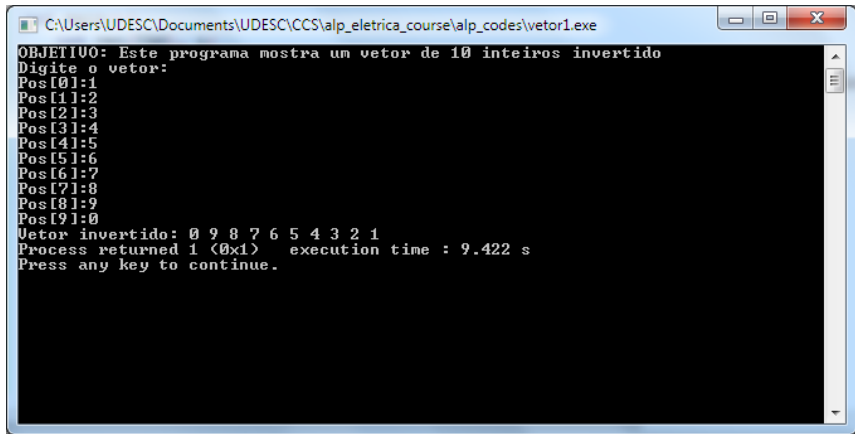
    for(i=0; i<TAM; i++) {
        printf("\nPos[%d]:",i);
        scanf("%d", &vet[i]);
    }

    printf("Vetor invertido: ");
    for(i=TAM-1; i>=0; i--)
        printf("%d ", vet[i]);

    return 1;
}
```

Estruturas homogêneas

Vetores



```

CAUsers\UDESC\Documents\UDESC\CCS\alp_eletrica_course\alp_codes\vetor1.exe
OBJETIVO: Este programa mostra um vetor de 10 inteiros invertido
Digite o vetor:
Pos[0]:1
Pos[1]:2
Pos[2]:3
Pos[3]:4
Pos[4]:5
Pos[5]:6
Pos[6]:7
Pos[7]:8
Pos[8]:9
Pos[9]:0
Vetor invertido: 0 9 8 7 6 5 4 3 2 1
Process returned 1 (0x1)   execution time : 9.422 s
Press any key to continue.
  
```

Estruturas homogêneas

Exercícios

1. Faça um algoritmo que leia um vetor de N posições (de inteiros) e inverta (fisicamente) o mesmo antes de exibí-lo.

Estruturas homogêneas

Exercícios

1. Faça um algoritmo que leia um vetor de N posições (de inteiros) e inverta (fisicamente) o mesmo antes de exibí-lo.
2. Faça um algoritmo que leia um vetor de N posições (de inteiros) e some seus elementos.

Estruturas homogêneas

Exercícios

1. Faça um algoritmo que leia um vetor de N posições (de inteiros) e inverta (fisicamente) o mesmo antes de exibí-lo.
2. Faça um algoritmo que leia um vetor de N posições (de inteiros) e some seus elementos.
3. Faça um algoritmo que leia um vetor de N posições (de inteiros) e some os elementos das posições pares e o produto dos elementos das posições ímpares.

Estruturas homogêneas

Exercícios

1. Faça um algoritmo que leia um vetor de N posições (de inteiros) e inverta (fisicamente) o mesmo antes de exibí-lo.
2. Faça um algoritmo que leia um vetor de N posições (de inteiros) e some seus elementos.
3. Faça um algoritmo que leia um vetor de N posições (de inteiros) e some os elementos das posições pares e o produto dos elementos das posições ímpares.
4. Faça um algoritmo que leia um vetor de N posições (de inteiros) e apresente o valor do maior e do menor elemento contido nele.

Estruturas homogêneas

Exercícios

1. Faça um algoritmo que leia um vetor de N posições (de inteiros) e inverta (fisicamente) o mesmo antes de exibí-lo.
2. Faça um algoritmo que leia um vetor de N posições (de inteiros) e some seus elementos.
3. Faça um algoritmo que leia um vetor de N posições (de inteiros) e some os elementos das posições pares e o produto dos elementos das posições ímpares.
4. Faça um algoritmo que leia um vetor de N posições (de inteiros) e apresente o valor do maior e do menor elemento contido nele.
5. Faça um algoritmo que leia dois vetores de N posições cada um (de inteiros) e some seus elementos nas posições correspondentes.

Estruturas homogêneas

Exercícios

1. Faça um algoritmo que leia um vetor de N posições (de inteiros) e inverta (fisicamente) o mesmo antes de exibí-lo.
2. Faça um algoritmo que leia um vetor de N posições (de inteiros) e some seus elementos.
3. Faça um algoritmo que leia um vetor de N posições (de inteiros) e some os elementos das posições pares e o produto dos elementos das posições ímpares.
4. Faça um algoritmo que leia um vetor de N posições (de inteiros) e apresente o valor do maior e do menor elemento contido nele.
5. Faça um algoritmo que leia dois vetores de N posições cada um (de inteiros) e some seus elementos nas posições correspondentes.

Estruturas homogêneas

Exercícios: Conjuntos

1. Faça um algoritmo que, dado um conjunto A contendo N números inteiros e um outro número inteiro X ; verifique se $X \in A$.

Estruturas homogêneas

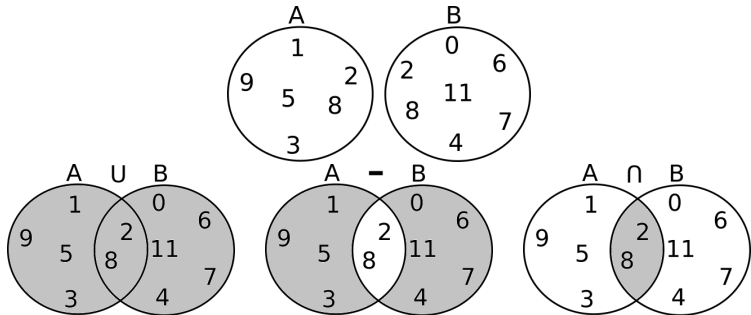
Exercícios: Conjuntos

1. Faça um algoritmo que, dado um conjunto A contendo N números inteiros e um outro número inteiro X ; verifique se $X \in A$.
2. **Programa-desafio!** Faça um algoritmo que leia dois conjuntos (A e B) contendo M e N números inteiros respectivamente e encontre a união, interseção e a diferença.

Estruturas homogêneas

Exercícios: Conjuntos

1. Faça um algoritmo que, dado um conjunto A contendo N números inteiros e um outro número inteiro X ; verifique se $X \in A$.
2. **Programa-desafio!** Faça um algoritmo que leia dois conjuntos (A e B) contendo M e N números inteiros respectivamente e encontre a união, interseção e a diferença.



Estruturas homogêneas

Exercícios: strings

1. Faça um algoritmo que leia uma string e conte a quantidade de vogais.

Estruturas homogêneas

Exercícios: strings

1. Faça um algoritmo que leia uma string e conte a quantidade de vogais.
2. Faça um algoritmo que leia uma string e apresente a quantidade letras e dígitos.

`isalpha(X)` retorna true se X é uma letra do alfabeto

`isdigit(X)` retorna true se X é um dígito

`isalnum(X)` retorna true se X é ou uma letra do alfabeto ou um dígito

Estruturas homogêneas

Exercícios: strings

1. Faça um algoritmo que leia uma string e conte a quantidade de vogais.
2. Faça um algoritmo que leia uma string e apresente a quantidade letras e dígitos.

`isalpha(X)` retorna true se X é uma letra do alfabeto

`isdigit(X)` retorna true se X é um dígito

`isalnum(X)` retorna true se X é ou uma letra do alfabeto ou um dígito

3. Faça um algoritmo que leia uma string e determine se ela é palíndrome (igual se lida de trás pra frente).

Estruturas homogêneas

Exercícios: strings

1. Faça um algoritmo que leia uma string e conte a quantidade de vogais.
2. Faça um algoritmo que leia uma string e apresente a quantidade letras e dígitos.

`isalpha(X)` retorna true se X é uma letra do alfabeto
`isdigit(X)` retorna true se X é um dígito

`isalnum(X)` retorna true se X é ou uma letra do alfabeto ou um dígito

3. Faça um algoritmo que leia uma string e determine se ela é palíndrome (igual se lida de trás pra frente).
4. Leia uma string *FONTE* e uma segunda string *ALVO*. Faça um algoritmo que conte quantas vezes *ALVO* aparece em *FONTE*. Exemplo:
FONTE = “o rato roeu a roupa do rei de roma” & *ALVO* = “ro” → “ro”
 aparece 3 vezes na frase.

Estruturas homogêneas

Exercícios: strings

1. Faça um algoritmo que leia uma string e conte a quantidade de vogais.
2. Faça um algoritmo que leia uma string e apresente a quantidade letras e dígitos.

`isalpha(X)` retorna true se X é uma letra do alfabeto
`isdigit(X)` retorna true se X é um dígito

`isalnum(X)` retorna true se X é ou uma letra do alfabeto ou um dígito

3. Faça um algoritmo que leia uma string e determine se ela é palíndrome (igual se lida de trás pra frente).
4. Leia uma string *FONTE* e uma segunda string *ALVO*. Faça um algoritmo que conte quantas vezes *ALVO* aparece em *FONTE*. Exemplo:
 FONTE = “o rato roeu a roupa do rei de roma” & ALVO = “ro” → “ro”
 aparece 3 vezes na frase.
5. Faça um algoritmo que leia uma string e remova as vogais e os espaços em branco. Exemplo: “universidade do estado de santa catarina” →
 “nvrsdddstdsntctrn”

Estruturas homogêneas

Exercícios: strings

1. Faça um algoritmo que leia uma string e conte a quantidade de vogais.
2. Faça um algoritmo que leia uma string e apresente a quantidade letras e dígitos.

`isalpha(X)` retorna true se X é uma letra do alfabeto
`isdigit(X)` retorna true se X é um dígito

`isalnum(X)` retorna true se X é ou uma letra do alfabeto ou um dígito

3. Faça um algoritmo que leia uma string e determine se ela é palíndrome (igual se lida de trás pra frente).
4. Leia uma string *FONTE* e uma segunda string *ALVO*. Faça um algoritmo que conte quantas vezes *ALVO* aparece em *FONTE*. Exemplo:
 FONTE = “o rato roeu a roupa do rei de roma” & ALVO = “ro” → “ro”
 aparece 3 vezes na frase.
5. Faça um algoritmo que leia uma string e remova as vogais e os espaços em branco. Exemplo: “universidade do estado de santa catarina” →
 “nvrsdddstdsntctrn”

Estruturas homogêneas

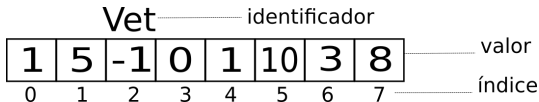
Vetores Multi-dimensionais

- Vetores multi-dimensionais representam uma variação dos vetores unidimensionais.

Estruturas homogêneas

Vetores Multi-dimensionais

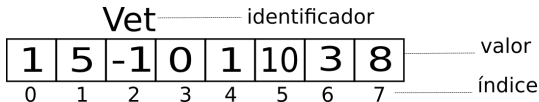
- Vetores multi-dimensionais representam uma variação dos vetores unidimensionais.
- Em um vetor unidimensional, temos um conjunto de valores homogêneos associados a um identificador único.



Estruturas homogêneas

Vetores Multi-dimensionais

- Vetores multi-dimensionais representam uma variação dos vetores unidimensionais.
- Em um vetor unidimensional, temos um conjunto de valores homogêneos associados a um identificador único.

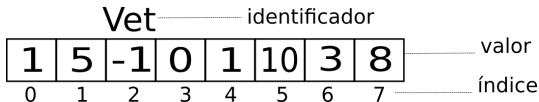


- No caso de vetores multi-dimensionais temos um vetor homogêneo do tipo vetor homogêneo. Ou seja, cada posição do vetor armazena um outro vetor de um certo tipo de dados.

Estruturas homogêneas

Vetores Multi-dimensionais

- Vetores multi-dimensionais representam uma variação dos vetores unidimensionais.
- Em um vetor unidimensional, temos um conjunto de valores homogêneos associados a um identificador único.



- No caso de vetores multi-dimensionais temos um vetor homogêneo do tipo vetor homogêneo. Ou seja, cada posição do vetor armazena um outro vetor de um certo tipo de dados.
- No exemplo abaixo declaramos um vetor de 4 posições, que armazena um vetor de 8 inteiros **em cada** posição.

```
int mat[4][8];
```

Estruturas homogêneas

Vetores Multi-dimensionais

```
int mat[4][8];  
mat[2][3] = 0;
```

Mat ———— identificador

	0	1	2	3	4	5	6	7	índice
0	1	5	-1	0	1	10	3	8	valores
1	1	5	-1	0	1	10	3	8	
2	1	5	-1	0	1	10	3	8	
3	1	5	-1	0	1	10	3	8	

Estruturas homogêneas

Vetores Multi-dimensionais

```
int mat[4][4];
```

1	2	3	4	[0][0]	[0][1]	[0][2]	[0][3]
5	6	7	8	[1][0]	[1][1]	[1][2]	[1][3]
9	10	11	12	[2][0]	[2][1]	[2][2]	[2][3]
13	14	15	16	[3][0]	[3][1]	[3][2]	[3][3]

Estruturas homogêneas

Vetores Multi-dimensionais

```
#include <stdio.h>

int main() {
    int mat[50][50], M, N, l, c;

    scanf("%d %d", &M, &N);
    for(l=0; l<M; l++)
        for(c=0; c<N; c++)
            scanf("%d",&mat[l][c]);

    for(l=0; l<M; l++) {
        for(c=0; c<N; c++)
            printf("%d ", mat[l][c]);
        printf("\n");
    }

    return 1;
}
```

Estruturas homogêneas

Vetores Multi-dimensionais

1. Faça um programa que apresente a matriz transposta (mat^T) de uma matriz quadrada mat de ordem M .

Estruturas homogêneas

Vetores Multi-dimensionais

1. Faça um programa que apresente a matriz transposta (mat^T) de uma matriz quadrada mat de ordem M .
2. Faça um programa que apresente o maior e o menor elemento de uma matriz quadrada de ordem M .

Estruturas homogêneas

Vetores Multi-dimensionais

1. Faça um programa que apresente a matriz transposta (mat^T) de uma matriz quadrada mat de ordem M .
2. Faça um programa que apresente o maior e o menor elemento de uma matriz quadrada de ordem M .
3. Faça um programa que apresente a soma dos elementos de uma matriz quadrada de ordem M .

Estruturas homogêneas

Vetores Multi-dimensionais

1. Faça um programa que apresente a matriz transposta (mat^T) de uma matriz quadrada mat de ordem M .
2. Faça um programa que apresente o maior e o menor elemento de uma matriz quadrada de ordem M .
3. Faça um programa que apresente a soma dos elementos de uma matriz quadrada de ordem M .
4. Faça um programa que apresente a soma dos elementos em cada linha de uma matriz $M \times N$.

Estruturas homogêneas

Vetores Multi-dimensionais

1. Faça um programa que apresente a matriz transposta (mat^T) de uma matriz quadrada mat de ordem M .
2. Faça um programa que apresente o maior e o menor elemento de uma matriz quadrada de ordem M .
3. Faça um programa que apresente a soma dos elementos de uma matriz quadrada de ordem M .
4. Faça um programa que apresente a soma dos elementos em cada linha de uma matriz $M \times N$.
5. Faça um programa que, dada uma matriz quadrada de ordem M , apresente a soma dos elementos da diagonal principal e o produto dos elementos da diagonal secundária.

Estruturas homogêneas

Vetores Multi-dimensionais

1. Faça um programa que apresente a soma de duas matrizes mat_A e mat_B ; ambas de ordem $M \times N$.

Estruturas homogêneas

Vetores Multi-dimensionais

1. Faça um programa que apresente a soma de duas matrizes mat_A e mat_B ; ambas de ordem $M \times N$.
2. Faça um programa que apresente a soma dos elementos em cada coluna de uma matriz de ordem $M \times N$.

Estruturas homogêneas

Vetores Multi-dimensionais

1. Faça um programa que apresente a soma de duas matrizes mat_A e mat_B ; ambas de ordem $M \times N$.
2. Faça um programa que apresente a soma dos elementos em cada coluna de uma matriz de ordem $M \times N$.
3. **Programa-desafio!** Faça um programa que, dada uma matriz mat_A (ordem $M \times N$) e outra matriz mat_B (ordem $P \times Q$), apresente a matriz mat_C onde $mat_C = mat_A \times mat_B$.

DICA! Para que o produto de matriz seja possível, o número de colunas da primeira matriz precisa necessariamente ser igual ao número de linhas da segunda matriz: $N = P$ e a matriz resultante terá ordem $M \times Q$.

Sub-Rotinas

- Algoritmos complexos apresentam sequências de código que se repetem em vários trechos do programa

Sub-Rotinas

- Algoritmos complexos apresentam sequências de código que se repetem em vários trechos do programa
- E ainda, múltiplos programas distintos frequentemente utilizam rotinas similares, obrigando a *reprogramação*

Sub-Rotinas

- Algoritmos complexos apresentam sequências de código que se repetem em vários trechos do programa
- E ainda, múltiplos programas distintos frequentemente utilizam rotinas similares, obrigando a *reprogramação*
- Solução: sub-rotinas

Sub-Rotinas

- Algoritmos complexos apresentam sequências de código que se repetem em vários trechos do programa
- E ainda, múltiplos programas distintos frequentemente utilizam rotinas similares, obrigando a *reprogramação*
- Solução: sub-rotinas
- Sub-rotinas representam outra forma de controle de fluxo: abordagem “dividir em sub-partes”

Sub-Rotinas

- Algoritmos complexos apresentam sequências de código que se repetem em vários trechos do programa
- E ainda, múltiplos programas distintos frequentemente utilizam rotinas similares, obrigando a *reprogramação*
- Solução: sub-rotinas
- Sub-rotinas representam outra forma de controle de fluxo: abordagem “dividir em sub-partes”
- Algoritmo deixa de ser um bloco único de execução para se tornar um processo modular (integração de sub-partes).

Sub-Rotinas

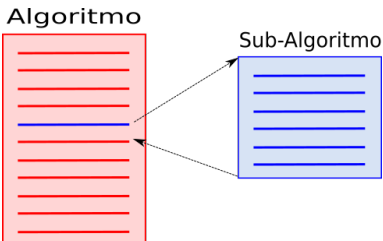
- Uma sub-rotina (ou sub-algoritmo), na prática, é um novo algoritmo (que pode conter os mesmos elementos) executado separadamente, tanto no tempo como no uso dos recursos computacionais (memória, dispositivos E/S)

Sub-Rotinas

- Uma sub-rotina (ou sub-algoritmo), na prática, é um novo algoritmo (que pode conter os mesmos elementos) executado separadamente, tanto no tempo como no uso dos recursos computacionais (memória, dispositivos E/S)
- Um programa que utiliza sub-rotinas pára a sua execução no momento que a sub-rotina inicia a execução, e só retoma a execução após esta encerrar e devolver o controle para o programa (estado de espera)

Sub-Rotinas

- Uma sub-rotina (ou sub-algoritmo), na prática, é um novo algoritmo (que pode conter os mesmos elementos) executado separadamente, tanto no tempo como no uso dos recursos computacionais (memória, dispositivos E/S)
- Um programa que utiliza sub-rotinas pára a sua execução no momento que a sub-rotina inicia a execução, e só retoma a execução após esta encerrar e devolver o controle para o programa (estado de espera)



Sub-Rotinas

Declaração de sub-rotinas

```
<tipo> <identificador> ( )  
{  
    <comandos>  
}
```

Sub-Rotinas

Declaração de sub-rotinas

```
<tipo> <identificador> ( )
{
    <comandos>
}
```

```
int subrotina ( )
{
    int x, y;
    scanf("%d", &x);
    y = x + 1;
    return y;
}
```

Sub-Rotinas

```

1  #include <stdio.h>
2
3  int  LeInt()
4  {
5      int  valor;
6      printf (" Digite um numero: ");
7      scanf ("%d", &valor);
8      return  valor;
9  }
10
11 int  main()
12 {
13     int  x, y;
14     x = LeInt();
15     y = LeInt();
16     printf ("Valores  lidos : %d %d", x, y);
17     return  1;
18 }
```

Sub-Rotinas

Escopo Global vs. Escopo Local

- Denomina-se **escopo** ao espaço de tempo no qual um determinado identificador existe em um programa

Sub-Rotinas

Escopo Global vs. Escopo Local

- Denomina-se **escopo** ao espaço de tempo no qual um determinado identificador existe em um programa

Escopo Global equivale ao período total da execução do programa

Sub-Rotinas

Escopo Global vs. Escopo Local

- Denomina-se **escopo** ao espaço de tempo no qual um determinado identificador existe em um programa

Escopo Global equivale ao período total da execução do programa
Identificadores declarados nesse escopo são reconhecíveis em qualquer ponto do programa

Sub-Rotinas

Escopo Global vs. Escopo Local

- Denomina-se **escopo** ao espaço de tempo no qual um determinado identificador existe em um programa

Escopo Global equivale ao período total da execução do programa
Identificadores declarados nesse escopo são reconhecíveis em qualquer ponto do programa

Escopo Local Identificadores são visíveis apenas durante o período de execução do referido escopo (p.ex. sub-rotina)

Sub-Rotinas

Escopo Global vs. Escopo Local

```

1  #include <stdio.h>
2
3  int Subrotina() {
4      ....
5      x = 0; // ERRO! Variavel x nao existe neste escopo
6      ...
7  }
8
9  int main() {
10     int x; // escopo local dentro de main()
11     ...
12     Subrotina();
13
14     return 1;
15 }
```

Sub-Rotinas

Escopo Global vs. Escopo Local

```

1  #include <stdio.h>
2
3  int x; // escopo global visível em qualquer ponto do programa
4
5  int Subrotina() {
6      ...
7      x = 0; // variável x existe neste escopo
8      ...
9  }
10
11 int main() {
12     ...
13     x = 1; // variável x existe neste escopo
14     ...
15     Subrotina();
16
17     return 1;
18 }
```

Sub-Rotinas

Escopo Global vs. Escopo Local

- Um mesmo identificador pode ser declarado múltiplas vezes em escopos locais distintos

Sub-Rotinas

Escopo Global vs. Escopo Local

- Um mesmo identificador pode ser declarado múltiplas vezes em escopos locais distintos
- Se isso ocorrer, cada identificador significa uma variável distinta (apesar de compartilharem o mesmo nome)

Sub-Rotinas

Escopo Global vs. Escopo Local

- Um mesmo identificador pode ser declarado múltiplas vezes em escopos locais distintos
- Se isso ocorrer, cada identificador significa uma variável distinta (apesar de compartilharem o mesmo nome)

```

1  #include <stdio.h>
2
3  int Subrotina() {
4      int x = 0; // variavel x existe apenas no escopo Subrotina()
5      ...
6  }
7
8  int main() {
9      int x = 1; // variavel x existe apenas no escopo main()
10     ...
11     Subrotina();
12     ...
13     return 1;
14 }
```

Sub-Rotinas

Escopo Global vs. Escopo Local

- Um mesmo identificador pode ser declarado múltiplas vezes em escopos de níveis distintos

Sub-Rotinas

Escopo Global vs. Escopo Local

- Um mesmo identificador pode ser declarado múltiplas vezes em escopos de níveis distintos
- Se isso ocorrer, cada identificador será referenciado pelo seu escopo mais próximo

Sub-Rotinas

Escopo Global vs. Escopo Local

- Um mesmo identificador pode ser declarado múltiplas vezes em escopos de níveis distintos
- Se isso ocorrer, cada identificador será referenciado pelo seu escopo mais próximo
- O operador (::) pode ser utilizado se o escopo mais alto precisa ser referenciado

Sub-Rotinas

Escopo Global vs. Escopo Local

- Um mesmo identificador pode ser declarado múltiplas vezes em escopos de níveis distintos
- Se isso ocorrer, cada identificador será referenciado pelo seu escopo mais próximo
- O operador (::) pode ser utilizado se o escopo mais alto precisa ser referenciado

Sub-Rotinas

Escopo Global vs. Escopo Local

```

1  #include <stdio.h>
2
3  int x; // variavel global
4
5  int Subrotina() {
6      int x = 0; // variavel local a Subrotina()
7
8      ...
9
10     ::x = 2; // uso da variavel global
11 }
12
13 int main() {
14     ...
15     x = 1; // uso da variavel global
16     ...
17 }
```

Sub-Rotinas

Tipos de Retorno

- Todos os tipos de dados suportados pela linguagem podem ser usados como retorno de uma função, inclusive vetores

Sub-Rotinas

Tipos de Retorno

- Todos os tipos de dados suportados pela linguagem podem ser usados como retorno de uma função, inclusive vetores
- Existe ainda um tipo especial **void** que funciona como um tipo nulo (sem retorno)

Sub-Rotinas

Tipos de Retorno

- Todos os tipos de dados suportados pela linguagem podem ser usados como retorno de uma função, inclusive vetores
- Existe ainda um tipo especial **void** que funciona como um tipo nulo (sem retorno)
- Funções *void* não retornam valores

Sub-Rotinas

```

1  #include <stdio.h>
2
3  void Interface () {
4      system("cls ");
5      printf ("+-+-----+\\n");
6      printf ("|          Programa desenvolvido por Rogerio Eduardo da Silva (2017)          |\\n");
7      printf ("+-+-----+\\n");
8      printf ("|          OBJETIVO: Apresentar o conceito de sub-rotinas          |\\n");
9      printf ("+-+-----+\\n");
10 }
11
12 int LeInt() {
13     int valor;
14     printf ("Digite um numero: ");
15     scanf("%d", &valor);
16     return valor;
17 }
18
19 int main() {
20     Interface ();
21     printf ("Valores lidos : %d %d", LeInt(), LeInt ());
22     return 1;
23 }

```

Sub-Rotinas

Comando return

- O comando **return** encerra a execução de uma sub-rotina retornando um resultado para o comando chamador

Sub-Rotinas

Comando return

- O comando **return** encerra a execução de uma sub-rotina retornando um resultado para o comando chamador
- É válido o retorno de expressões desde que o resultado seja de um tipo compatível com o declarado na função: **return A+B;**

Sub-Rotinas

Comando return

- O comando **return** encerra a execução de uma sub-rotina retornando um resultado para o comando chamador
- É válido o retorno de expressões desde que o resultado seja de um tipo compatível com o declarado na função: **return A+B;**
- Não é obrigatório que o comando chamador utilize o retorno fornecido pela função

Sub-Rotinas

Comando return

- O comando **return** encerra a execução de uma sub-rotina retornando um resultado para o comando chamador
- É válido o retorno de expressões desde que o resultado seja de um tipo compatível com o declarado na função: **return A+B;**
- Não é obrigatório que o comando chamador utilize o retorno fornecido pela função
 - `int x = LeInt();`
 - `LeInt();`

Sub-Rotinas

Comando return

- O comando **return** encerra a execução de uma sub-rotina retornando um resultado para o comando chamador
- É válido o retorno de expressões desde que o resultado seja de um tipo compatível com o declarado na função: **return A+B;**
- Não é obrigatório que o comando chamador utilize o retorno fornecido pela função
 - `int x = LeInt();`
 - `LeInt();`
- É permitido o uso de mais de um ponto de retorno em uma função (p.ex. dentro de um comando condicional) desde que se garanta que ao menos um deles seja sempre executado

Sub-Rotinas

Comando return

- O comando **return** encerra a execução de uma sub-rotina retornando um resultado para o comando chamador
- É válido o retorno de expressões desde que o resultado seja de um tipo compatível com o declarado na função: **return A+B;**
- Não é obrigatório que o comando chamador utilize o retorno fornecido pela função
 - `int x = LeInt();`
 - `LeInt();`
- É permitido o uso de mais de um ponto de retorno em uma função (p.ex. dentro de um comando condicional) desde que se garanta que ao menos um deles seja sempre executado
- Funções do tipo *void* também podem ter um retorno sem valor: **return;** para garantir o encerramento forçado da sub-rotina

Sub-Rotinas

Comando return

- O comando **return** encerra a execução de uma sub-rotina retornando um resultado para o comando chamador
- É válido o retorno de expressões desde que o resultado seja de um tipo compatível com o declarado na função: **return A+B;**
- Não é obrigatório que o comando chamador utilize o retorno fornecido pela função
 - `int x = LeInt();`
 - `LeInt();`
- É permitido o uso de mais de um ponto de retorno em uma função (p.ex. dentro de um comando condicional) desde que se garanta que ao menos um deles seja sempre executado
- Funções do tipo *void* também podem ter um retorno sem valor: **return;** para garantir o encerramento forçado da sub-rotina
- Qualquer comando declarado após a execução do comando de retorno não será executado

Sub-Rotinas

Comando return

```
1 void Subrotina1() {
2     ...
3
4     return ;
5
6     x++; // ERRO! Nunca sera executado este comando
7 }
8
9 int Subrotina2() {
10     ...
11
12     if (X > Y)
13         return A+B;
14
15     return A;
16 }
```

Sub-Rotinas

Parâmetros

- Uma das principais vantagens do uso de sub-rotinas é que estas podem ser projetadas para uso em múltiplos casos, através do recebimento de **parâmetros externos**

Sub-Rotinas

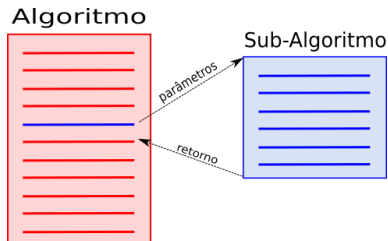
Parâmetros

- Uma das principais vantagens do uso de sub-rotinas é que estas podem ser projetadas para uso em múltiplos casos, através do recebimento de **parâmetros externos**
- Um parâmetro nada mais é do que uma variável local (ou lista de variáveis) declarada dentro dos parênteses da sub-rotina que é inicializada por um valor fornecido pelo programa chamador

Sub-Rotinas

Parâmetros

- Uma das principais vantagens do uso de sub-rotinas é que estas podem ser projetadas para uso em múltiplos casos, através do recebimento de **parâmetros externos**
- Um parâmetro nada mais é do que uma variável local (ou lista de variáveis) declarada dentro dos parênteses da sub-rotina que é inicializada por um valor fornecido pelo programa chamador



Sub-Rotinas

Parâmetros

```

1  #include <stdio.h>
2
3  int Soma(int x, int y) {
4      return x + y;
5  }
6
7  int main() {
8      int a, b;
9      scanf("%d %d", &a, &b);
10     printf ("A+B = %d\n", Soma(a,b));
11     printf ("A+5 = %d", Soma(a,5));
12     return 1;
13 }
```

Sub-Rotinas

Parâmetros Formais vs Parâmetros Reais

Parâmetros Formais são as variáveis declaradas dentro da função que irão receber o valor fornecido pelo programa chamador

Sub-Rotinas

Parâmetros Formais vs Parâmetros Reais

Parâmetros Formais são as variáveis declaradas dentro da função que irão receber o valor fornecido pelo programa chamador

Parâmetros Reais são os valores fornecidos pelo programa chamador no momento da chamada da sub-rotina

Sub-Rotinas

Parâmetros Formais vs Parâmetros Reais

Parâmetros Formais são as variáveis declaradas dentro da função que irão receber o valor fornecido pelo programa chamador

Parâmetros Reais são os valores fornecidos pelo programa chamador no momento da chamada da sub-rotina

Importante ! Um parâmetro real não obrigatoriamente precisa ser uma variável, podendo também ser uma expressão, uma constante ou até mesmo um valor literal

Sub-Rotinas

Tipos de Passagem de Parâmetros

Por Valor quando apenas o conteúdo do parâmetro real é atribuído ao parâmetro formal

Sub-Rotinas

Tipos de Passagem de Parâmetros

Por Valor quando apenas o conteúdo do parâmetro real é atribuído ao parâmetro formal
Em outras palavras: o parâmetro formal recebe uma cópia do valor proveniente do parâmetro real (alterar o conteúdo de uma não afeta o conteúdo da outra)

Sub-Rotinas

Tipos de Passagem de Parâmetros

Por Valor quando apenas o conteúdo do parâmetro real é atribuído ao parâmetro formal

Em outras palavras: o parâmetro formal recebe uma cópia do valor proveniente do parâmetro real (alterar o conteúdo de uma não afeta o conteúdo da outra)

Por Referência quando o endereço de memória do parâmetro real é copiado no parâmetro formal

Sub-Rotinas

Tipos de Passagem de Parâmetros

Por Valor quando apenas o conteúdo do parâmetro real é atribuído ao parâmetro formal

Em outras palavras: o parâmetro formal recebe uma cópia do valor proveniente do parâmetro real (alterar o conteúdo de uma não afeta o conteúdo da outra)

Por Referência quando o endereço de memória do parâmetro real é copiado no parâmetro formal

Em outras palavras: as duas variáveis compartilham o mesmo endereço de memória (alterar o conteúdo de uma, altera o conteúdo da outra)

Sub-Rotinas

Tipos de Passagem de Parâmetros

Por Valor quando apenas o conteúdo do parâmetro real é atribuído ao parâmetro formal

Em outras palavras: o parâmetro formal recebe uma cópia do valor proveniente do parâmetro real (alterar o conteúdo de uma não afeta o conteúdo da outra)

Por Referência quando o endereço de memória do parâmetro real é copiado no parâmetro formal

Em outras palavras: as duas variáveis compartilham o mesmo endereço de memória (alterar o conteúdo de uma, altera o conteúdo da outra)

Importante ! Apenas variáveis podem ser fornecidas como parâmetro real quando da passagem por referência

Sub-Rotinas

Tipos de Passagem de Parâmetros

Por Valor quando apenas o conteúdo do parâmetro real é atribuído ao parâmetro formal

Em outras palavras: o parâmetro formal recebe uma cópia do valor proveniente do parâmetro real (alterar o conteúdo de uma não afeta o conteúdo da outra)

Por Referência quando o endereço de memória do parâmetro real é copiado no parâmetro formal

Em outras palavras: as duas variáveis compartilham o mesmo endereço de memória (alterar o conteúdo de uma, altera o conteúdo da outra)

Importante ! Apenas variáveis podem ser fornecidas como parâmetro real quando da passagem por referência

Importante 2! O parâmetro real deve incluir um símbolo (&) enquanto que o parâmetro formal deve conter um (*)

Sub-Rotinas

Tipos de Passagem de Parâmetros

Por Valor quando apenas o conteúdo do parâmetro real é atribuído ao parâmetro formal

Em outras palavras: o parâmetro formal recebe uma cópia do valor proveniente do parâmetro real (alterar o conteúdo de uma não afeta o conteúdo da outra)

Por Referência quando o endereço de memória do parâmetro real é copiado no parâmetro formal

Em outras palavras: as duas variáveis compartilham o mesmo endereço de memória (alterar o conteúdo de uma, altera o conteúdo da outra)

Importante ! Apenas variáveis podem ser fornecidas como parâmetro real quando da passagem por referência

Importante 2! O parâmetro real deve incluir um símbolo (&) enquanto que o parâmetro formal deve conter um (*)

Sub-Rotinas

Tipos de Passagem de Parâmetros

```

1  #include <stdio.h>
2
3  void Func1(int x, int y) {
4      x = 2 * x;
5      y = -1;
6  }
7
8  void Func2(int *x, int *y) {
9      *x = 2 * (*x);
10     *y = -1;
11 }
12
13 int main() {
14     int a, b;
15     scanf("%d %d", &a, &b);
16     Func1(a, b);
17     printf("X = %d Y = %d\n", a, b);
18     Func2(&a, &b);
19     printf("X = %d Y = %d\n", a, b);
20     return 1;
21 }

```

Sub-Rotinas

Assinatura de Funções

- Em programas com muitas funções ocorre o transtorno de que o ponto de início da execução (**main()**) acaba ficando no final do código-fonte

Sub-Rotinas

Assinatura de Funções

- Em programas com muitas funções ocorre o transtorno de que o ponto de início da execução (**main()**) acaba ficando no final do código-fonte
- Solução? Uso de assinaturas

Sub-Rotinas

Assinatura de Funções

- Em programas com muitas funções ocorre o transtorno de que o ponto de início da execução (**main()**) acaba ficando no final do código-fonte
- Solução? Uso de assinaturas
- É chamado de *assinatura* de uma função à declaração de existência da referida função, o que permite que a sua descrição seja feita em qualquer outro local do código (p.ex. após a descrição do `main()`)

Sub-Rotinas

Assinatura de Funções

- Em programas com muitas funções ocorre o transtorno de que o ponto de início da execução (**main()**) acaba ficando no final do código-fonte
- Solução? Uso de assinaturas
- É chamado de *assinatura* de uma função à declaração de existência da referida função, o que permite que a sua descrição seja feita em qualquer outro local do código (p.ex. após a descrição do `main()`)
- Na declaração da assinatura da função apenas os tipos de dados envolvidos (parâmetros e retorno) precisam ser declarados

Sub-Rotinas

Assinatura de Funções

- Em programas com muitas funções ocorre o transtorno de que o ponto de início da execução (**main()**) acaba ficando no final do código-fonte
- Solução? Uso de assinaturas
- É chamado de *assinatura* de uma função à declaração de existência da referida função, o que permite que a sua descrição seja feita em qualquer outro local do código (p.ex. após a descrição do `main()`)
- Na declaração da assinatura da função apenas os tipos de dados envolvidos (parâmetros e retorno) precisam ser declarados

```
int Subrotina(int, int);
```

Sub-Rotinas

Assinatura de Funções

```
1  #include <stdio.h>
2
3  int Soma(int, int );
4
5  int main() {
6      int a, b;
7      scanf("%d %d", &a, &b);
8      printf ("A+B = %d", Soma(a, b));
9      return 1;
10 }
11
12 int Soma(int x, int y) {
13     int resultado = x + y;
14     return resultado;
15 }
```

Sub-Rotinas

Programação Modular

- Consiste em se dividir o código-fonte em múltiplos arquivos e então compilar todos os arquivos para só então efetuar a linkedição em um programa executável final

Sub-Rotinas

Programação Modular

- Consiste em se dividir o código-fonte em múltiplos arquivos e então compilar todos os arquivos para só então efetuar a linkedição em um programa executável final
- O processo de compilação é um pouco diferente em Windows e em Linux

Sub-Rotinas

Programação Modular

- Consiste em se dividir o código-fonte em múltiplos arquivos e então compilar todos os arquivos para só então efetuar a linkedição em um programa executável final
- O processo de compilação é um pouco diferente em Windows e em Linux
- A forma de escrita dos arquivos segue a mesma ideia em qualquer sistema operacional:

Sub-Rotinas

Programação Modular

- Consiste em se dividir o código-fonte em múltiplos arquivos e então compilar todos os arquivos para só então efetuar a linkedição em um programa executável final
- O processo de compilação é um pouco diferente em Windows e em Linux
- A forma de escrita dos arquivos segue a mesma ideia em qualquer sistema operacional:
- O código-fonte é separado em arquivos de cabeçalho ***.h** (ou *header files*) e arquivos fonte ***.c** (ou *source files*)

Sub-Rotinas

Programação Modular

- Consiste em se dividir o código-fonte em múltiplos arquivos e então compilar todos os arquivos para só então efetuar a linkedição em um programa executável final
- O processo de compilação é um pouco diferente em Windows e em Linux
- A forma de escrita dos arquivos segue a mesma ideia em qualquer sistema operacional:
- O código-fonte é separado em arquivos de cabeçalho ***.h** (ou *header files*) e arquivos fonte ***.c** (ou *source files*)
- Um arquivo cabeçalho contém a lista de assinaturas das funções utilizadas pelo programa e o arquivo fonte contém a implementação dessas funções

Sub-Rotinas

Programação Modular

bases.h

```

1  #ifndef BASES
2  #define BASES
3
4  int  BinDec(char [15]);
5  char [15] DecBin(int);
6  int  HexDec(char [15]);
7  char[15] DecHex(int);
8  int  OctDec(char [15]);
9  char [15] DecOct(int);
10
11 #endif

```

Sub-Rotinas

Programação Modular

bases.c

```

1  #include "bases.h"
2  #include <math.h>
3  #include <string.h>
4
5  int BinDec(char numero[15]) {
6      int i, resultado = 0, tam = strlen(numero);
7      for (i=0; i < tam; i++)
8          if (numero[i] == '1')
9              resultado += pow(2, tam - i - 1);
10     return resultado;
11 }
12
13 char [15] DecBin(int numero) {
14     char resultado [15];
15     strcpy (resultado, "");
16     while (numero > 1)
17     {
18         strcpy (temp, numero % 2 == 1 ? "1" : "0");
19         strcpy (resultado, strcat (temp, resultado ));
20         numero /= 2;
21     }
22     strcpy (temp, numero == 1 ? "1" : "0");
23     strcpy (resultado, strcat (temp, resultado ));
24     return resultado;
25 }
26
27 ...

```

Sub-Rotinas

Programação Modular

programa.c

```
1  #include "bases.h"
2
3  int main() {
4      int x;
5      scanf("%d", &x);
6      printf ("%d dec = %s bin\n", x, DecBin(x)); // DecBin declarada em bases.h
7      return 1;
8  }
```

Sub-Rotinas

Programação Modular em Windows

- Para se compilar um programa modular em CodeBlocks para Windows, utiliza-se o conceito de projeto

Sub-Rotinas

Programação Modular em Windows

- Para se compilar um programa modular em CodeBlocks para Windows, utiliza-se o conceito de projeto
- Um projeto permite a compilação de múltiplos arquivos de uma vez só. Cria-se um novo projeto no menu **File | New | Project** e escolhe-se a opção **Console Application**

Sub-Rotinas

Programação Modular em Windows

- Para se compilar um programa modular em CodeBlocks para Windows, utiliza-se o conceito de projeto
- Um projeto permite a compilação de múltiplos arquivos de uma vez só. Cria-se um novo projeto no menu **File | New | Project** e escolhe-se a opção **Console Application**
- Criar e incluir no projeto cada um dos arquivos conforme descrito anteriormente

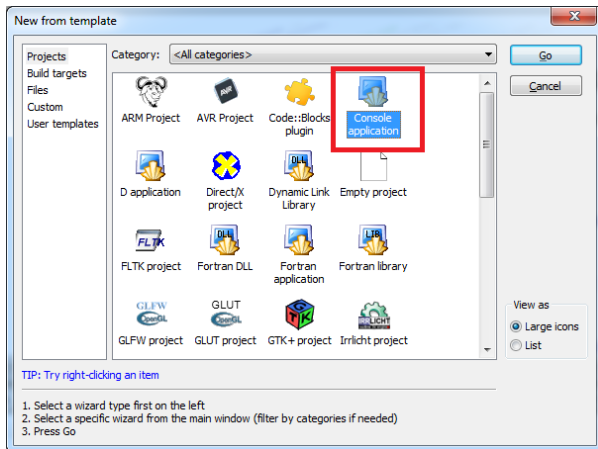
Sub-Rotinas

Programação Modular em Windows

- Para se compilar um programa modular em CodeBlocks para Windows, utiliza-se o conceito de projeto
- Um projeto permite a compilação de múltiplos arquivos de uma vez só. Cria-se um novo projeto no menu **File | New | Project** e escolhe-se a opção **Console Application**
- Criar e incluir no projeto cada um dos arquivos conforme descrito anteriormente
- A compilação então segue da maneira convencional: `build + run`

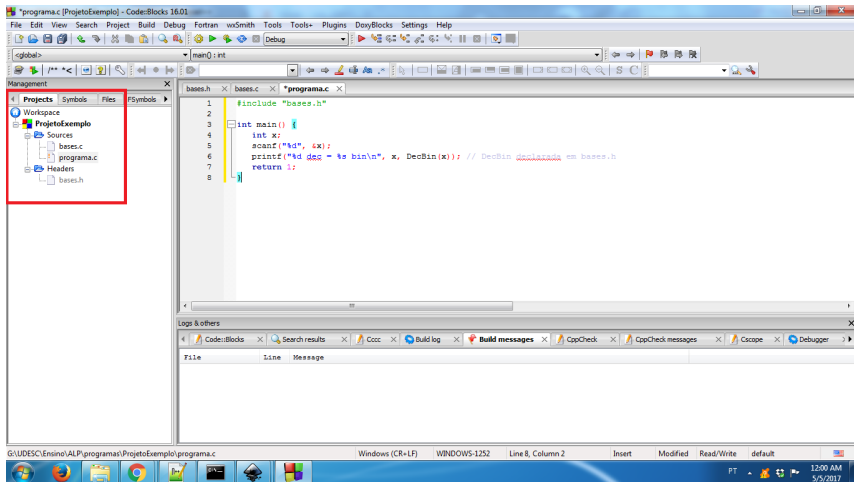
Sub-Rotinas

Programação Modular em Windows



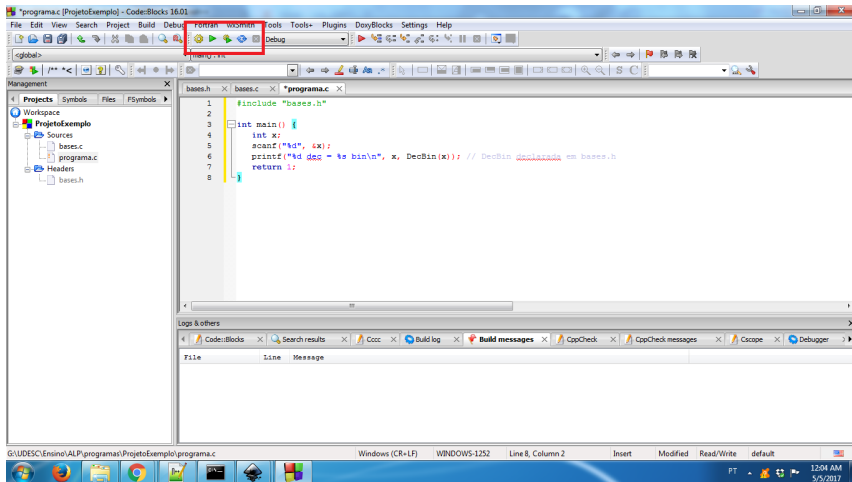
Sub-Rotinas

Programação Modular em Windows



Sub-Rotinas

Programação Modular em Windows



Unidade 05

Projeto Final

Previsão: 08 horas/aula

Próxima Seção

Em Construção