**Aula # 1**

**Tema 1 –** Conceito sobre Linguagens de Programação

**Sumário:** Introdução e breves considerações.

A diciplina de T.L.P justifica-se enquanto suporte técnico-prático a análise e desenvolvimento de programas informáticos.

A área de informática é composto pelos seguintes cursos:

* Técnico de informática;
* Técnico de Gestão de Sistemas Informáticos;
* Técnico de informática e Multimídea.

Área funcional

* Empresas de desenvolvimento e comercialização.

Finalidades

* Montar e manter sistemas informáticos;
* Programar e gerar aplicações informáticas;
* Instalar e gerir sistemas operativos e de redes.

Serviços feitos no dia-a-dia na área de informática.

Os exercícios a apresentar têm dois objetivos fundamentais:

* Fixar as técnicas de programação;
* Adquirir estrututra logica de raciocinio cada vez mais apurada para a pogramação.

Históricamente as linguagens de programação estão diretamente relacionadas ao surgimento das primeiras máquinas inteligentes, por meio delas, o intuito sempre foi transformar as sequências lógicas em resolução de problemas.

Diferentes llinguagens de programação para resolver diferentes tipos de problemas.

Tarefa

1 – Dê uma definição para a disciplina de TLP.

Compreende os conceitos e boas práticas para a criação de um programa de computador utilizando as técnicas e recursos como variáveis, tipos de dados, modularização, estrutura de controlo, funções, procedimentos, etc.

**Aula # 2**

**Tema 1 –** Conceito sobre Linguagens de Programação

**Sumário:** Conceitos básicos sobre softwares.

Programar é conhecer os fundamentos, a lógica, que está na base de todas as linguagens de programação. Com isso, é possível utilizar qualquer linguagem de programação.

O conceito de software refere-se à unidade lógica do computador.

Software é o conjunto de ordens e instruções que tornam possível ao computador realizar determinadas tarefas.

Estas instruções codificadas em uma linguagem compreensível pelo computador constituem programas, qua tal como nossos pensamentos e ideias, orientam o nosso comportamento, controlam e regulam o funcionamento do computador.

**Regras sintáticas** consistem num conjunto de normas a seguir que indicam como se deve estruturar o código, ou seja, como se deve construir o código.

Os softwares podem ser divididos em software de sistema, software de aplicação e softwares de programação.

**Linguagem de programação** é um conjunto de símbolos (comandos, identificadores, caracteres ASCII, etc.) e regras de sintaxe que permitem a construção de sentenças que descrevem de forma precisa ações compreensíveis e executáveis para o computador.

***LINGUAGEM DE PROGRAMAÇÃO*** *= SÍMBOLOS + REGRAS DE SINTAXE*

Uma linguagem de programação é uma notação formal para descrição de algoritmos que serão executados por um computador.

Como todas as notações formais, uma linguagem de programação tem dois componentes: **Sintaxe** e **Semântica**.

A sintaxe consiste em um conjunto de regras formais, que especificam a composição de programas a partir de letras, dígitos, e outros símbolos. Por exemplo, regras de sintaxe podem especificar que cada parêntese aberto em uma expressão aritmética deve corresponder a um parêntese fechado, e que dois comandos quaisquer devem ser separados por um ponto-e-vírgula.

As regras de semântica especificam o “significado” de qualquer programa, sintaticamente válido, escrito na linguagem.

**Tipos de Linguagens de Programação**

As várias linguagens de programação podem ser classificadas em três tipos distintos:

Linguagens Máquina;

Linguagem Assembly (simbólica);

Linguagens de Alto Nível.

**Linguagem de Baixo Nível e Linguagens de Alto Nível**

***Linguagem Máquina*** consiste num conjunto de números binários (sequencias de 0 e 1) que são entendidas pela Unicdad Central de Processamento (CPU). Têm a ver com carateristicas intrísecas do processador do computador.

Única compreendida pelo computador. Específica de cada computador.

***Linguagem Assembly*** Consiste numa linguagem muito próxima da linguagem de máquina, em que cada opcode (código de operação) se associa a uma palavra (*mnemotécnica – palavra que nos ajuda a lembrar de uma ação realizada pelo comando*).

Utilizam mnemônicos para representar instruções elementares

***Linguagem de Alto Nível*** são linguagens que permitem a especificação de instruções para o processador numa forma abreviada, onde cada instrução representa várias instruções em linguagem de máquina.

Utiliza instruções próximas da linguagem humana de forma a facilitar o raciocínio.

São fáceis de aprender, requerem menos tempo para escrever, têm melhor documentação.

Ex.:

Uso Científico : Fortran

Propósito Geral : Pascal, C, Basic

Uso Comercial : Cobol, Clipper

Uso específico : Lisp, Prolog

Visuais : Delphi, Visual Basic

Tarefa

1 – Escreva o ranking das linguagens de programação atualmente.

2 – Define linguagens de programação.

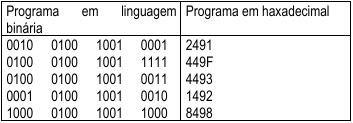
3 – Dê exemplos de linguagens de programação de alto nível.

4 - Classifique as linguagens em 2 grupos, de propósito geral e linguagens específicas.

**LINGUAGEM DE MÁQUINA**

O tipo mais primitivo de linguagem de programação é a linguagem que o computador entende diretamente, isto é, as instruções que podem ser diretamente executadas pelo hadware, isto é, pela CPU. É a linguagem de máquina, que foi utilizada pela primeira geração de programadores. A única verdadeira linguagem de programação é a linguagem de máquina, mas, para a maioria das pessoas, a linguagem de máquina é ininteligível.

Um programa em linguagem de máquina é uma longa seqüência de números, alguns dos quais representam instruções e outros, os dados a serem manipulados pelas instruções.



Para escrever um programa em linguagem de máquina, o programador deve conhecer todas as instruções disponíveis para aquela máquina e seus respectivos códigos de operação e formatos, assim como os registradores da CPU disponíveis e os endereços das células de memória onde serão armazenadas as instruções e os dados.

Um programa real em linguagem de máquina, pode conter milhares de instruções, o que é uma tarefa extremamente tediosa e difícil, pelos detalhes que precisam ser observados pelo programador.

Em um programa escrito em linguagem de máquina, cada instrução escrita pelo programador será individualmente executada, isto é, a cada instrução do programa corresponderá uma ação do computador.

A relação é portanto 1 para 1 - uma instrução do programa corresponde a uma operação do computador. Cada família de computadores possui sua própria linguagem de máquina.

Um programa em linguagem de máquina é dependente do computador ou seja, tendo sido escrito para um determinado computador, somente poderá ser executado em computadores da mesma família, que lhe sejam 100% compatíveis.

**LINGUAGEM ASSEMBLY**

Para os cientistas que desenvolveram o primeiro programa para traduzir instruções em código de máquina, qualquer linguagem mais fácil de compreender do que a linguagem de máquina teria sido considerada uma linguagem de alto nível.

Para tentar minimizar esta ineficiência (dificuldade de entendimento do significado dos números, todos semelhantes), foi desenvolvida, ainda para a primeira geração de computadores, uma linguagem que representasse as instruções por símbolos e não por números. A linguagem assenbly está mais próxima da linguagem humana, mas também não apresenta legibilidade nem portabilidade.

Esta linguagem simbólica foi denominada Linguagem de Montagem (Assembly Language).

Segundo os padrões dos dias de hoje, a linguagem assembly é uma linguagem de nível muito baixo, porque seus comandos correspondem, um a um, ao conjunto de instruções de uma CPU. Na verdade não existe apenas uma linguagem assembly. Cada tipo de CPU que tem um conjunto de instruções exclusivo tem sua própria linguagem assembly.

Para usar-se linguagem de montagem em um computador, é necessário que haja um meio de converter os símbolos alfabéticos utilizados no programa em código de máquina, de modo a poder ser compreendido pela CPU. O processo de conversão (também denominado tradução) é chamado de montagem e é realizado por um programa chamado Montador (Assembler). O montador lê cada instrução em linguagem de montagem e cria uma instrução equivalente em linguagem de máquina.

A linguagem assembly continua, apesar de ser de uso tedioso e complexo, a ser importante, porque dá aos programadores total controle sobre a CPU do computador. Além disso produz-se um código compacto, rápido e eficiente.

Vantagens:

• Os programas escritos apresentam grande velocidade de execução;

• Os programas gerados são mais compactos, portanto ocupam menos espaço em memória;

• programador tem à sua disposição todos os recursos da máquina, sem limitações da linguagem.

Desvantagens:

• Os programadores devem ter altíssimo nível técnico;

• Os programas levam muito tempo para serem desenvolvidos e apresentam grande dificuldade para alterações futuras;

• Os programas são muito específicos para cada microprocessador (não são portáveis).

Aula # 3

Tema 1 – Conceito sobre Linguagens de Programação

Sumário: Conceitos sobre compilador e interpretador (O processo de tradução).

Vimos que programas de computador não são escritos na linguagem que o computador entende (linguagem de máquina), mas sim em outras formas simbólicas de representar as instruções que o programador deseja que sejam realizadas. No entanto, verificamos que as máquinas continuam entendendo somente em binário e, por isso, sempre há a necessidade da conversão ou tradução de um programa em linguagem simbólica para outro, equivalente em linguagem numérica binária.

**PROBLEMA:** Como a linguagem de nível mais alto pode ser implementada em um computador, cuja linguagem é bastante diferente a de nível mais baixo?

**SOLUÇÃO:** Através da tradução de programas escritos em linguagens de alto nível para a linguagem de baixo nível do computador.

Um programa escrito por um programador (chamado código fonte) em uma linguagem de alto nível é um conjunto de instruções que é clara para programadores, mas não para computadores. Ou seja, os computadores entendem única e exclusivamente suas linguagens nativas, as linguagens de máquina. Programas em linguagem de alto nível, a exemplo dos programas escritos em linguagem de Montagem, também precisam ser traduzidos para linguagem de máquina para poderem ser submetidos ao computador e processados.

Um programa escrito em linguagem de alto nível é um simples texto, texto escrito usando programa do tipo editor de texto, posteriormente descodificado nas correspondentes instruções em linguagem de máquina.

Existem dois percursos para execução do programa escrito em linguagem de alto nível: interpretadores e compiladores.

**Interpretador** lê no programa cada instrução de cada vez, por cada insrução lida, verifica quais são as acções a tomar em linguagem de máquina e executa-as.

Um interpretador não cria nenhum outro programa.

Efetua a tradução a de uma linguagem de alto nível para linguagem de máquina da seguinte forma:

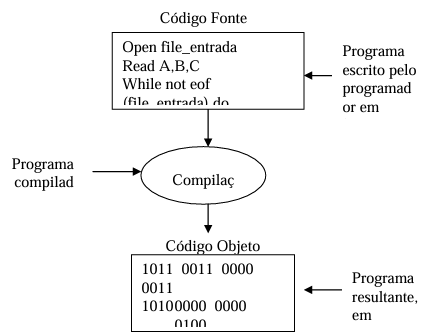
1 - Obtém próxima instrução do código-fonte em linguagem de alto nível;

2 - Traduz para as instruções correspondentes em linguagem de máquina;

3 - Executa as instruções em linguagem de máquina;

4 - Repete o passo 1 até o fim do programa.

**Compilador** lê um certo número de vezes o programa inicial em linguagem de alto nível (texto) e constrói um segundo programa equivalente ao primeiro mas escrito em linguagem de máquina.



Efetua a tradução de todo o código-fonte em linguagem de alto nível para as instruções correspondentes em linguagem de máquina, gerando o código-objeto do programa. Em seguida é necessário o uso de um outro programa (Link-Editor) que é responsável pela junção de diversos códigos-objeto em um único programa executável.



***Possibilidades de Erros no Programa:***

*Erros de Compilação:* Erros de digitação e de uso da sintaxe da linguagem.

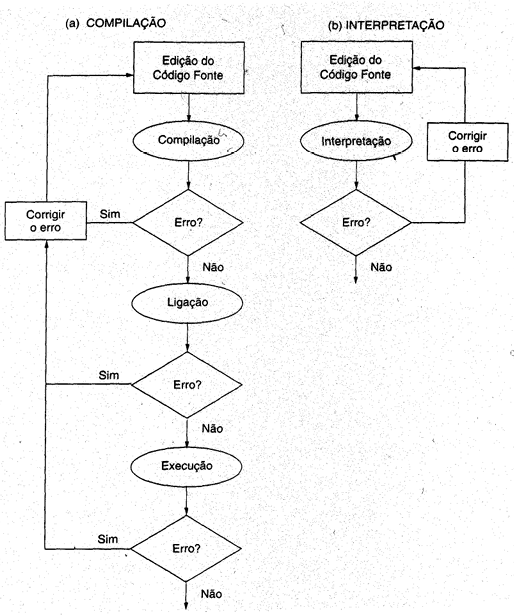
*Erros de Link-Edição:* Erro no uso de bibliotecas de sub-programas necessárias ao programa principal.

*Erros de Execução:* Erro na lógica do programa (algoritmo).

Compilação e Interpretação - comparação

Sempre que houver duas opções, haverá vantagens e desvantagens para cada uma delas (pois se assim não fosse, a que apresentasse sempre desvantagem seria abandonada).

Fluxograma Compilação x Interpretação



1º Tempo de execução: No método de interpretação, cada vez que o programa for executado, haverá compilação, ligação e execução de cada um dos comandos. No método de Compilação, o tempo de execução do programa é reduzido, porque todos os passos preliminares (compilação e ligação) foram previamente cumpridos.

2º Consumo de memória No método de interpretação, o interpretador é um programa geralmente grande e que precisa permanecer na memória durante todo o tempo que durar a execução do programa, pois um programa necessita do interpretador para ter traduzidos cada um dos seus comandos, um a um, até o término de sua execução (o interpretador somente é descarregado depois do término da execução do programa). No método de compilação, o compilador é carregado e fica na memória apenas durante o tempo de compilação, depois é descarregado; o ligador é carregado e fica na memória apenas durante o tempo de ligação, depois é descarregado. Essas são funções realizadas pelo programador e executadas apenas durante o desenvolvimento do programa. Quando o usuário for executar o programa, apenas o módulo de carga (código executável) é carregado e fica na memória durante a execução. Desta forma, vemos que o método de interpretação acarreta um consumo de memória muito mais elevado durante a execução do programa.

3º Repetição de interpretação: No método de compilação, um programa é compilado e ligado apenas uma vez, e na hora da execução é carregado apenas o módulo de carga, que é diretamente executável. No método de interpretação, cada programa terá que ser interpretado toda vez que for ser executado. Outro aspecto é que, em programas contendo loops, no método de interpretação as partes de código pertencentes ao loop serão várias vezes repetidas e terão que ser interpretadas tantas vezes quantas o loop tiver que ser percorrido. No método de compilação, a tradução do código do loop se faz uma única vez, em tempo de compilação e ligação. Estas características levam a um maior consumo de tempo no método de interpretação, que é portanto mais lento.

4º Desenvolvimento de programas e depuração de erros: No método de compilação, a identificação de erros durante a fase de execução fica sempre difícil, pois não há mais relação entre comandos do código fonte e instruções do executável. No método de interpretação, cada comando é interpretado e executado individualmente, a relação entre código fonte e executável é mais direta e o efeito da execução (certa ou errada) é direta e imediatamente sentido. Quando a execução de um comando acarreta erro, quase sempre o erro pode ser encontrado no comando que acabou de ser executado. Assim, o interpretador pode informar o erro, indicando o comando ou variável causador do problema. Essa característica, que é a rigor a maior vantagem do método de interpretação sob o ponto de vista do usuário, faz com que esse método seja escolhido sempre que se pretende adotar uma linguagem que vá ser usada por não-profissionais ou para a programação mais expedita. Por exemplo, o método de interpretação é usado pela maioria dos Basic (uma linguagem projetada para ser usada por iniciantes), e por todas as linguagens típicas de usuário como dBase, Access, Excel, etc.

**Montador** Efetua a tradução de linguagem de montagem (Assembly) para a linguagem de máquina.

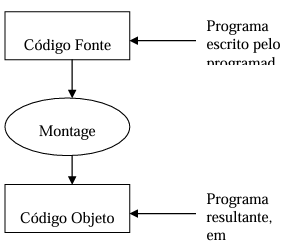
1 - Obtém próxima instrução do Assembly;

2 - Traduz para as instruções correspondentes em linguagem de máquina;

3 - Executa as instruções em linguagem de máquina;

4 - Repete o passo 1 até o fim do programa.

É o tipo de tradução mais rápido e simples que existe e é realizada por um programa denominado montador (assembler). Como o nome já explica, a montagem é realizada para traduzir um programa em linguagem de montagem para seu equivalente em linguagem binária, executável. Cada ordem do programador é convertida diretamente em uma única ordem de máquina. Neste processo o programa escrito em linguagem de montagem, chamado de código fonte, é examinado, instrução por instrução, e, em seguida, é convertido (traduzido) para um outro programa em linguagem binária de máquina, denominado código objeto.



**Programa Fonte, Programa Tradutor e Programa Objeto**

**Programa fonte** é o texto que nós escrevemos, utilizando um editor de texto e segundo as regras sintáticas utilizadas na linguagem de programação com que vamos programar.

**Programa Tradutor** são programas que traduzem o programa fonte em linguagem de máquina, utilizando para tal o programa objeto criado a partir do programa fonte. Com a utilização dos programas tradutores obtemos um programa que pode ser executado directamente pela máquina (computador).

**Programa Objeto** é o programa que obtemos após termos feito a compilação do programa fonte, utilizando um compilador, ou seja, após ter sido traduzido o programa fonte.

Podemos dizer, dessa forma, que o processo de programação consiste nos seguintes passos:



**LINGUAGEM DE ALTO NÍVEL**

A linguagem de alto nível é uma linguagem de fácil aprendizado, apresenta uma melhor legibilidade, além de ser independente da máquina, isto é, apresenta portabilidade. Da mesma forma que os programas com instruções de máquina, os programas feitos em uma linguagem de alto nível, também requerem uma conversão para instruções de máquinas. Este processo é denominado compilação. Normalmente, a conversão de um simples comando Cobol ou C, redunda em dezenas de instruções de máquina, diferentemente de um programa em linguagem de montagem, no qual cada instrução implica uma única instrução de máquina. Veja abaixo um trecho de um programa em linguagem C (linguagem de alto nível).

Vantagens:

• programador não se preocupa com aspectos muito técnicos, permitindo que ele se preocupe com o problema a ser resolvido;

• Permitem rápido desenvolvimento de novos sistemas;

• Permitem que os programadores aprendam mais rapidamente, programem com mais eficiência e façam alterações mais facilmente em programas anteriormente escritos;

• Maior portabilidade, ou seja, facilidade de transferência de programas para máquinas diferentes.

Desvantagens:

• Os programas (código) ocupam mais espaço de memória;

• A estrutura da linguagem tira parte da liberdade do programador e pode dificultar a implementação de instruções mais técnicas, aproveitando menos os recursos da máquina.

• A velocidade de execução dos programas tende a ser mais lenta.

Foram desenvolvidas diversas linguagens de programação, buscando afastar-se do modelo centrado no computador. Essas linguagens foram estruturadas buscando refletir melhor os processos humanos de solução de problemas. Essas linguagens orientadas a problema são também chamadas linguagens de alto nível, por serem afastadas do nível de máquina.

As primeiras linguagens foram FORTRAN (1957), usada basicamente para manipulação de fórmulas;

ALGOL (1958), para manipulação de algoritmos;

COBOL (1959), para processamento comercial e ainda hoje bastante usada, especialmente em computadores de grande porte (mainframes) em bancos.

Nas décadas de 60 e 70, podemos citar Pascal, a primeira linguagem de alto nível estruturada; BASIC, linguagem criada para facilitar a programação por não-profissionais; e ADA, linguagem para processamento em tempo real criada sob encomenda do DoD (Department of Defense norte-americano) e ainda hoje a única linguagem aceita para programas escritos sob encomenda do DoD.

Na década de 80, surgiu o C e depois o C++ (com suporte a objetos), que estão entre as linguagens mais utilizadas hoje.

Cada nova linguagem criada visa atingir níveis de abstração mais altos, isto é, afastam cada vez mais o programador do nível de máquina. Se por um lado essas novas linguagens facilitam muito o trabalho do programadores (e reduzem sua necessidade de conhecer o hardware da máquina), elas cobram um alto preço em termos de desempenho (isto é, são cada vez mais lentas, ao consumir cada vez mais ciclos de máquina e espaço em memória).

Esse aumento de exigência ao poder de processamento dos computadores é compensado (e desta forma se faz menos notado) pelo aumento acelerado do poder de processamento dos novos chips (exemplificado pela chamada Lei de Moore, que afirma que o poder de processamento dos chips dobra a cada 18 meses) e pelos avanços na arquitetura dos computadores. Dentre as importantes tendências atuais, citamos as linguagens de manipulação de bancos de dados (como dBase, Clipper, FoxPro, Paradox, Access, etc) e as linguagens visuais, como o Visual Basic, Visual C e Delphi.

Aula # 4

Tema 2 – Introdução à algoritmia e algorítmos fundamentais

Sumário: Algoritmos.

Ao final deste capítulo você deverá ser capaz de:

• Definir algoritmo

• Descrever suas principais características

• Criar algoritmos utilizando diferentes formas de representação

**Algoritmo** é uma sequência de passos (operações) necessárias para realizar uma dada tarefa.

**Algoritmo** é um conjunto finito de regras, bem definidas, para a solução de um problema em um tempo finito e com um número finito de passos.

A defnição de como os passos de um algoritmo serão executados é uma implementação do algoritmo.

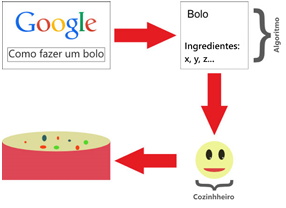
Resumindo, algoritmo é o que deve ser feito e implementação é o como deve ser feito.

Serve como modelo para programas, pois sua linguagem é intermediaria a linguagem humana e as linguagens de programação, sendo então, uma boa ferramenta na validação da lógica de tarefas a serem automatizadas.

Os algoritmos, servem para representar a solução de qualquer problema, mas no caso de processamento de dados, eles devem seguir as regras basicas de programação para que sejam compatíveis com as linguagens de programação.

Carateristicas do algoritmo:

* Deve ser deterministico
* Deve atingir uma solução em tempo finito.
* Deve considerar todas as situações particulares do problema, não contendo instruções ambíguas.
* Deve ter capacidade de comunicar com o exterior.



Os algoritmos podem ser representados por 4 formas:

* Linguagem corrente;
* Fluxograma;
* Diagrama de chappin;
* Pseudocódigo.

Algoritmos não se aprendem:

Copiando algoritmo

Estudando algoritmo

Algoritmos so se aprendem:

Construindo algoritmos

Testando algoritmos

Aula # 5

Tema 2 – Introdução à algoritmia e algorítmos fundamentais

Sumário: Linguagem corrente (Representação dos algoritmos).

Linguagem corrente é a forma de represenar os algoritmos indicando os passos a seguir na linguagem do dia-a-dia.

Um comando é uma ordem ou instrução dada a um computador ou qualquer outra máquina automatizada.

Aula # 4

Tema 2 – Introdução à algoritmia e algorítmos fundamentais

Sumário: Fluxograma (Representação dos algoritmos).

Fluxograma é um diagrama de fluxo que consiste na representação gráfica do algoritmo usando um conjunto de figuras geométricas simples, ligadas através da seta.

Um fluxograma é uma representação gráfica de um algoritmo que utiliza símbolos de forma a demonstrar os processos neste realizado.



Aula # 4

Tema 2 – Introdução à algoritmia e algorítmos fundamentais

Sumário: Diagrama de chappin (Representação dos algoritmos).

Aula # 5

Tema 2 – Introdução à algoritmia e algorítmos fundamentais

Sumário: Concepção de um programa.

Um programa pode ser visto como algo que transforma informação.



Estamos interessados em desenvolver algoritmos computacionais. Para isso, utilizaremos um modelo de programação. Um modelo de programação fornece idéias e conceitos para nos ajudar a criar algoritmos.

A concepção de um programa obedece uma sequencia de 4 fases:

1º Formulação do problema: estudar o problema de forma a encontrar um modelo que o represente corretamente.

2º Resolução do problema: obter um algoritmo que conduz a solução do problema.

3º Tradução: traduzir o algoritmo numa linguagem de programação escolhida.

4º Teste: efetur conjunto de testes com o objetivo de verificar se o programa cumpre os objetivos pretendidos.

Aula # 6

Tema 2 – Introdução à algoritmia e algorítmos fundamentais

Sumário: Pseudocodigo (Representação dos algoritmos).

**Pseudocódigo** é uma forma de representação de algoritmos que se assemelha a lingua gens de programação mas que utiliza a língua nativa do utilizador de forma a ser facilmente entendida por quem não tem quaisquer conhecimentos da sintaxe de uma linguagem de programação.

Os programadores cuja língua nativa é português, costumam referir-se ao pseudocódigo como Portugol, também conhecido por “Português Estruturado”.

Um comando é uma ordem ou instrução dada a um computador ou qualquer outra máquina automatizada.

**Constantes e variáveis**

A declaração de variáveis, ou constantes, consiste no processo em que o compilador é "avisado"da sua existência para que, de cada vez se menciona o nome da variável/constante, esta seja utilizada.

A partir do momento declaramos uma variável ou constante, o endereço da memória RAM que foi reservado estará disponível através do seu nome. Sempre que for referenciado receita no código, o valor da variável será retornado.

**Variáveis**

Espaço reservado na memória do computador, no qual podemos guardar um determinado tipo de informação.

Na linguagem algorítmica (portugues estruturado) a atribuição de um valor a uma variável é utilizando o operador de atribuição.

**Tipos de dados**

Cada variavel é classificada segundo um determinado tipo de dado, de acordo com os valores que vão armazenar.

Os tipos de dados são: real, caracter, string, booleano, data time, inteiro.

**Tipo inteiro:** refere-se a números inteiros negativos e positivos pertencentes a uma determinada gama dependendo do computador.

**Tipo real:** define o conjunto dos números reais respeitando uma gama dependente do computador onde se está a executar o programa. decimal (163.17; 0.023), notação cientifica (1.6 E 3)

**Tipo caracter:** permite que todos os elementos sejam do tipo caacter individuais que podem ser representados por um computador. Código ASCII (American Standard Code for Information Interchange)) e o carater correspondente ao espaço vazio.

**Tipo boleano:** representa o conjunto de valores verdadeiros e falsos.

**Declaração de variáveis**

A declaração de variáveis é feita pelo tipo de dados da variável seguido do nome da variavel.

As variáveis devem ter nomes educativos das suas funções no programa.

**Constantes**

São valores atribuidos a um simbolo que não vão ser alterados ao longo da execução do programa. Constantes permitem armazenar valores imutáveis, ou seja, que não podem ser alte rados ao longo da execução de um programa.

**Nomenclatura**

Convencionalmente, o nome das constantes é escrito com letras maiúsculas para haver uma melhor distinção no código-fonte. Esta convenção não é obrigatória e não irá causar quaisquer erros durante a execução de um programa. Seguir esta convenção apenas torna mais clara a distinção entre variáveis e constantes dentro do código-fonte facilitando tanto a si, como programador, como a outros programadores que vejam o código do seu programa.

Variáveis, ao contrário das constantes, permitem o armazenamento de valores que podem ser alterados durante a execução de um programa. Geralmente são utilizadas para manter estados de algo e são fundamentais na programação.

Aula # 6

Sumário: Regras de nomenclatura das variáveis.

As regras de nomenclatura de variáveis podem variar conforme a linguagem de programação e as convenções de estilo adotadas.

Seguir essas regras e convenções ajuda a melhorar a legibilidade e a manutenção do código, facilitando o trabalho em equipe e reduzindo a possibilidade de erros.

1º Significado e Clareza: usar nomes descritivos e significativos que indiquem claramente o propósito da variável.

Exemplo: `idade`, `nomeCliente`, `quantidadeProdutos`.

2º Consistência de Estilo: seguir uma convenção de nomenclatura consistente, como camelCase, snake\_case, ou PascalCase, dependendo das normas da linguagem e do projeto.

Exemplo (camelCase): `numeroDePedidos`.

Exemplo (snake\_case): `numero\_de\_pedidos`.

Exemplo (PascalCase): `NumeroDePedidos`.

3º Comprimento Adequado: o nome deve ser suficientemente longo para ser descritivo, mas não excessivamente verboso.

 Exemplo: `totalVenda` em vez de `t` ou

`valorTotalDaVendaRealizadaDuranteOperacao`.

4º Evitar Palavras Reservadas: não usar palavras reservadas da linguagem de programação como nomes de variáveis.

Exemplo (em JavaScript): evite usar `var`, `function`, `class`.

5º Evitar Abreviações Ambíguas: usar abreviações bem conhecidas e evite aquelas que possam ser confusas.

   - Exemplo: `quantidade` em vez de `qtd`.

6º Usar Prefixos e Sufixos (quando apropriado): em algumas convenções, prefixos

como `is`, `has`, ou `can` são usados para booleanos.

   - Exemplo: `isAtivo`, `hasDesconto`.

7º Case Sensitivity: Muitas linguagens são sensíveis a maiúsculas e minúsculas, então seja consistente.

   - Exemplo: `dataInicio` e `datainicio` são variáveis diferentes em linguagens sensíveis a caso.

8º Evitar Nomes de Variáveis de Uma Letra: a menos que em contextos muito limitados (como em loops).

   - Exemplo: use `indice` em vez de `i`, exceto em loops curtos onde `i` é comumente aceito.

**Tarefa**

Explica o conceito de tutorial

Aula # 6

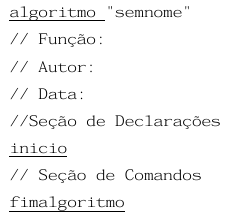
Tema 2 – Introdução à algoritmia e algorítmos fundamentais

Sumário: softwares para execução de algoritmos.

Fluxogramas (Diagramação)

VisualG

A linguagem VisuAlg é simples, seu objetivo é disponibilizar um ambiente de programação no idioma português do Brasil utilizando para tal o “Portugol” muito usado na bibliografia da área.



Entre as principais características estruturais da linguagem vale destacar:

• O VisuAlg permite apenas um comando por linha;

• Todas as palavras-chave do VisuAlg foram implementadas sem acentos, cedilha, etc.

• O VisuAlg não distingue maiúsculas e minúsculas no reconhecimento de palavras-chave e nomes de variáveis.

• É permitida a inclusão de comentários: qualquer texto precedido de "//" é ignorado, até se atingir o final da sua linha.

Portugol

Aula # 6

Tema 2 – Introdução à algoritmia e algorítmos fundamentais

Sumário: Operadores Aritméticos (Representação dos algoritmos).

Miniprojeto: Calculadora (fase 1)

Aula # 6

Tema 2 – Introdução à algoritmia e algorítmos fundamentais

Sumário: Instruções condicionais.

Aula # 6

Tema 2 – Introdução à algoritmia e algorítmos fundamentais

Sumário: Instruções cíclicas (repetição).

Conceito de ciclo, iteração, repetição.