Residência em Software

Aula 8
Programação Estruturada

Professores:

Alvaro Degas Coelho,
Edgar Alexander,
Esbel Thomas Valero,
Helder Conceição Almeida

INSTITUIÇÃO EXECUTORA: cepedi, UESC

COORDENADORA: MCTI FUTURO, Softex

APOIO: Ministério da Ciência, Tecnologia e Inovação, Governo Federal Brasil União e Reconstrução

Residência em Software

Programação Estruturada

- O conceito da programação estruturada teve seu início entre os anos 60 e 70, com o intuito de enfrentar os desafios com a complexidade crescente dos programas.
- As linguagens da época não ofereciam uma forma clara e organizada de escrever programas complexos.
- Em um artigo escrito em 1966, **Edsger W. Dijkstra** argumentou que o uso excessivo da instrução “**go to**” em programas dificultava a leitura, manutenção e entendimento do código, tornando os programas propensos a erros e falhas.

Residência em Software

Programação Estruturada

- Em 1968, **Niklaus Wirth** desenvolveu a linguagem de programação ALGOL W, que foi a primeira linguagem a introduzir formalmente o conceito de **blocos estruturados**.
- Um bloco estruturado agrupa um conjunto de instruções em um bloco, permitindo que eles sejam tratados como uma unidade lógica, conceito esse que se tornou fundamental para a programação estruturada, facilitando a leitura e a manutenção dos códigos desenvolvidos.
- Com a adoção crescente da programação estruturada, os programadores começaram a escrever código mais limpo, eficiente e fácil de manter, sendo um fator essencial para o desenvolvimento de sistemas mais robustos e confiáveis.

Residência em Software

Exemplo de programa em Basic usando goto

```
10 LET count = 1
20 PRINT "Contagem regressiva:"
30 PRINT count
40 LET count = count + 1
50 IF count <= 10 THEN GOTO 30
60 PRINT "Contagem finalizada!"
70 END
```

Residência em Software

Outro exemplo

```
10  REM RESOLVE EQUACAO DO SEGUNDO GRAU
20  READ A,B,C
30  IF A=0 THEN GOTO 400
40  LET D=B*B-4*A*C
50  IF D<0 THEN GOTO 420
60  PRINT "SOLUCAO"
70  IF D=0 THEN GOTO 200
80  PRINT "PRIMEIRA SOLUCAO",(-B+SQR(D))/(2*A)
90  PRINT "SEGUNDA SOLUCAO",(-B-SQR(D))/(2*A)
100  GOTO 20
200 PRINT "SOLUCAO UNICA",(-B)/(2*A)
300 GOTO 20
400 PRINT "A DEVE SER DIFERENTE DE ZERO"
```

```
410 GOTO 20
420 PRINT "NAO HA SOLUCOES REAIS"
430 GOTO 20

490 DATA 10,20,1241,123,22,-1
500 END
50  IF D<0 THEN GOTO 420
60  PRINT "SOLUCAO"
70  IF D=0 THEN GOTO 200
80  PRINT "PRIMEIRA SOLUCAO",(-B+SQR(D))/(2*A)
90  PRINT "SEGUNDA SOLUCAO",(-B-SQR(D))/(2*A)
100  GOTO 20
```

```
200 PRINT "SOLUCAO UNICA",(-B)/(2*A)
300 GOTO 20
400 PRINT "A DEVE SER DIFERENTE DE ZERO"
410 GOTO 20
420 PRINT "NAO HA SOLUCOES REAIS"
430 GOTO 20
490 DATA 10,20,1241,123,22,-1
500 END
```

Residência em Software

Funções e Procedimentos

- Funções e procedimentos são conceitos fundamentais em linguagens de programação que permitem a modularização e reutilização de código.
- Ambos são blocos de código que encapsulam um conjunto de instruções, mas têm algumas diferenças importantes em relação ao seu comportamento e propósito.

Residência em Software

Procedimentos

- Um procedimento, também conhecido como sub-rotina ou função sem retorno, é uma sequência de instruções nomeada que pode ser chamada (invocada) a partir de diferentes partes do programa para executar uma tarefa específica.

Residência em Software

Procedimentos

- As principais características de um procedimento são:
  - **Nome:**
    - Todo procedimento tem um nome exclusivo que o identifica dentro do programa.
  - **Parâmetros:**
    - Pode receber zero ou mais parâmetros (variáveis de entrada) que são utilizados no processamento dentro do procedimento.
  - **Corpo:**
    - Contém o conjunto de instruções que são executadas quando o procedimento é chamado.
  - **Chamada:**
    - É chamado por meio de sua identificação (nome), e a execução do programa avança para o bloco de código do procedimento. Ao finalizar o procedimento, a execução retorna para a instrução após a chamada.
  - **Não tem Retorno:**
    - Como não tem valor de retorno, o procedimento não produz nenhum resultado além das alterações nos parâmetros (caso existam) durante sua execução.

Residência em Software

Exemplo Procedimento

```
void mostra_media(float nota1, float nota2)
{
    float media;
    media = (nota1 + nota2)/2;
    cout << media;
}
```

Tipo: float
Nome: mostra_media
Parâmetros: nota1 e nota2
Corpo: todo conteúdo compreendido entre { e }.
Retorno: não existe

Residência em Software

Funções

- Uma função é semelhante a um procedimento, mas com uma diferença crucial: ela retorna um valor após seu processamento.

Residência em Software

Funções

- As principais características de uma função são:
  - **Nome:**
    - Toda função tem um nome exclusivo que o identifica dentro do programa.
  - **Parâmetros:**
    - Pode receber zero ou mais parâmetros (variáveis de entrada) que são utilizados no processamento dentro do procedimento.
  - **Corpo:**
    - Contém o conjunto de instruções que são executadas quando o procedimento é chamado.
  - **Chamada:**
    - É chamado por meio de sua identificação (nome), e a execução do programa avança para o bloco de código do procedimento. Ao finalizar o procedimento, a execução retorna para a instrução após a chamada.
  - **Retorno:**
    - Retorna um valor após sua execução, que pode ser utilizado em expressões ou atribuído a variáveis.

Residência em Software

Exemplo Função

```
float calcula_media(float nota1, float nota2)
{
    float media;
    media = (nota1 + nota2)/2;
    return media;
}
```

Tipo: float
Nome: calcula_media
Parâmetros: nota1 e nota2
Corpo: todo conteúdo compreendido entre { e }.
Retorno: valor da variável média

Residência em Software

Procedimentos e Funções em C++

Como na linguagem C++ funções e procedimentos são tratados da mesma forma, passaremos a chamar no restante dos slides, ambos como funções.

Residência em Software

Parâmetros

- Os parâmetros de um procedimento ou função são variáveis ou valores que são **passados para a função quando ela é chamada**.
- Eles permitem que os dados sejam enviados para a função, permitindo que ela realize operações ou cálculos com esses valores.
- Os parâmetros são definidos na declaração da função e **funcionam como variáveis locais dentro do escopo da função**.

Residência em Software

Parâmetros

- Deve-se evitar utilizar dentro das funções variáveis que não sejam criadas dentro das funções ou passadas por parâmetro. São as chamadas **variáveis globais**.
- A utilização de parâmetros está fortemente ligado a qualidade da função.
- A correta utilização dos parâmetros faz com que as funções sejam mais independentes e mais reutilizáveis.

Residência em Software

Exemplo do uso de parâmetros

Forma errada de usar uma função

```
int n1 = 5;    // variável global
int n2 = 8;    // variável global

int soma()
{
    int soma;
    soma = n1 + n2;
    return soma;
}
```

Forma correta de usar uma função

```
int soma(int n1, int n2)
{
    int soma;          // variável local
    soma = n1 + n2;
    return soma;
}
```

Residência em Software

Chamada de uma função

- A chamada de uma função, também conhecida como invocação de função, refere-se ao ato de executar o código contido em uma função específica durante a execução de um programa.
- Quando uma função é chamada, o controle do programa é transferido para o bloco de código da função, e as instruções dentro dela são executadas.
- Para chamar uma função, você precisa usar o nome da função seguido por parênteses que podem conter os argumentos (valores ou variáveis) que serão passados para a função, caso ela tenha parâmetros.

Residência em Software

Chamada de uma função

- A sintaxe geral para chamar uma função é a seguinte:

```
nome_da_funcao(argumento1, argumento2, ...)
```

Utilizando funções

```
int main(void){

    string nome;
    int anoNascimento;

    cout << "Digite o seu nome: ";
    cin >> nome;
    cumprimenta(nome);
    cout << " Tenha um bom curso.";
    cout << endl;
    cout << nome <<", em que ano voce nasceu? ";
    cin >> anoNascimento;
    cout << "Considerando que estamos em 2023, ou voce tem "
         << 2023-anoNascimento
         <<" anos, ou esta proximo de fazer."
         << endl;

    return 0;
}
```

Foi feita uma função que cumprimenta a pessoa conforme o horário do dia
Com “Bom dia” / “Boa tarde“/ “Boa noite”

Utilizando funções

```
// ------------------------------------------------------
//  Conforme o horário do dia, a função dá:
//           Bom dia / Boa tarde / Boa noite
//------------------------------------------------------
void cumprimenta(string nome)
{
    if ((horaAtual() >= 1) && (horaAtual() < 12))      // Manhã
    {
        cout << "Bom dia " << nome << ", " << endl;
    }
    else if ((horaAtual() >= 12) && (horaAtual() < 18)) // Tarde
    {
        cout << "Boa tarde " << nome << ", " << endl;
    }
    else                                                // Noite
    {
        cout << "Boa noite " << nome << ", " << endl;
    }
}
```

O procedimento ***cumprimenta*** precisa de uma outra função chamada ***horaAtual***, que retorna a hora atual.

Residência em Software

Utilizando funções

Esta é a função horaAtual. Apesar de não receber parâmetros, esta função não está utilizando variáveis globais, mas do sistema.

```
//---------------------------------------------------------
//  Retorna a hora do horario atual
// Exemplo: Se são 15:43:56h será retornado 15.
//---------------------------------------------------------
int horaAtual()
{
    time_t t = time(nullptr);
    tm* now =  localtime(&t);
    return now->tm_hour;          // retorna hh como inteiro
}
```

Residência em Software

Variável Global

- Uma variável global é uma variável declarada fora de qualquer função ou bloco específico dentro de um programa e, portanto, pode ser acessada e utilizada por todas as funções e blocos de código em todo o programa.

- Em outras palavras, ela tem um escopo que abrange todo o programa.

- Geralmente, as variáveis globais são declaradas no início do programa, fora de qualquer função, e estão disponíveis para serem utilizadas em qualquer lugar do código.

Residência em Software

Variável Global - Características

1. **Escopo Global:**
   - As variáveis globais têm um escopo que abrange todo o programa. Portanto, elas podem ser acessadas de qualquer função ou bloco.
2. **Acesso Universal:**
   - Como seu escopo é global, todas as funções e blocos podem ler e modificar o valor da variável global.
3. **Persistência:**
   - A variável global mantém seu valor entre as chamadas de função e persiste ao longo da execução do programa.
4. **Uso Cauteloso:**
   - Embora as variáveis globais ofereçam acesso universal, é necessário usá-las com cuidado, pois podem levar a problemas de legibilidade e depuração. O uso excessivo de variáveis globais pode tornar o código menos organizado e mais difícil de entender e manter.

Residência em Software

Variável Local

- Uma variável local é uma variável declarada dentro de uma função ou bloco específico e, portanto, só pode ser acessada e utilizada dentro desse escopo local.
- Ela não é visível nem acessível em outras partes do programa fora do escopo onde foi declarada.
- Isso significa que ela é "local" a uma função específica ou a um bloco de código.

Residência em Software

Características

1. **Escopo Local:**
   - As variáveis locais têm um escopo restrito ao bloco de código ou função onde foram declaradas. Elas não são visíveis em outras partes do programa.
2. **Acesso Limitado:**
   - As variáveis locais só podem ser lidas e modificadas dentro do escopo onde foram declaradas. Elas não podem ser acessadas fora desse escopo.
3. **Persistência Temporária:**
   - As variáveis locais são criadas quando o bloco ou função é executado e são destruídas quando a execução do bloco ou função é concluída. Portanto, elas têm uma vida útil temporária, e seus valores não persistem entre as chamadas de função.
4. **Boa Prática:**
   - O uso de variáveis locais é recomendado sempre que possível, pois ajuda a evitar problemas de dependências indesejadas entre diferentes partes do código e torna o programa mais modular e legível.

Residência em Software

Importante lembrar

Variáveis globais são acessíveis em todo o programa e persistem durante toda a execução

Variáveis locais têm um escopo limitado e uma vida útil temporária dentro de um bloco de código ou função específica.

A escolha adequada entre variáveis globais e locais depende das necessidades e requisitos específicos do programa.

Residência em Software

Parâmetros
Passagem por valor e por referência

- Existem duas formas em que os dados podem ser passados para dentro da função: uma é por valor e a outra por referência.
- Dizemos que uma informação é passada **por valor** quando ao invocarmos a função colocamos como parâmetro uma variável ou uma constante, a qual será **copiada** para uma **variável local à função**, sendo esta então utilizada em seu corpo e mantendo o conteúdo da **variável original inalterado**.
- Já na passagem **por referência** indicamos que será passado para a variável local o mesmo **endereço de memória utilizado pela variável passada por parâmetro**, ou seja, tanto a variável local quanto a variável externa a função vão utilizar o mesmo endereço. Desta forma ao fazer uma alteração na variável local referenciada no parâmetro, altera-se também o valor da variável externa.

Residência em Software

Passagem por valor

```cpp
#include <iostream>

// Função que recebe um parâmetro por valor
void passagemPorValor(int x) {
    x = x * 2; // Modificando o valor local do parâmetro
    std::cout << "Valor dentro da função: " << x << std::endl;
}

int main() {
    int numero = 5;
    std::cout << "Valor antes da função: " << numero << std::endl;
    passagemPorValor(numero);
    std::cout << "Valor depois da função: " << numero << std::endl;
    return 0;
}
```

SAIDA:
Valor antes da função: 5
Valor dentro da função: 10
Valor depois da função: 5

Residência em Software

Passagem por referência

```cpp
#include <iostream>

// Função que recebe um parâmetro por referência
void passagemPorReferencia(int &x) {
    x = x * 2; // Modificando o valor original através da referência
    std::cout << "Valor dentro da função: " << x << std::endl;
}

int main() {
    int numero = 5;
    std::cout << "Valor antes da função: " << numero << std::endl;
    passagemPorReferencia(numero);
    std::cout << "Valor depois da função: " << numero << std::endl;
    return 0;
}
```

SAÍDA:
Valor antes da função: 5
Valor dentro da função: 10
Valor depois da função: 10

Residência em Software

Modularização
Programação TOP-DOWN e BOTTOM-UP

- São duas abordagens diferentes para projetar e desenvolver algoritmos ou programas de computador.

- Elas representam estilos contrastantes de resolução de problemas, com ênfase em diferentes aspectos do desenvolvimento de software.

Residência em Software

Programação TOP-DOWN

- A programação top-down é uma abordagem de desenvolvimento que se concentra em dividir um problema maior em subproblemas menores e mais gerenciáveis.

- Ela começa com uma visão geral do problema e, em seguida, divide-o em subproblemas menores, que também são divididos em subproblemas ainda menores, e assim por diante, até que cada subproblema seja simples o suficiente para ser facilmente resolvido.

- Essa abordagem geralmente envolve o uso de funções ou módulos para tratar cada subproblema.

Residência em Software

Processo TOP-DOWN

- O processo pode ser descrito da seguinte forma:

a. Compreender o problema geral.
b. Identificar os subproblemas.
c. Resolver cada subproblema individualmente.
d. Combinar as soluções dos subproblemas para resolver o problema geral.

Residência em Software

Programação BOTTOM-UP

- A abordagem bottom-up funciona de maneira oposta à abordagem top-down.
- Inicialmente, inclui o projeto das partes mais fundamentais que são então combinadas para fazer o módulo de nível superior.
- Esta integração de submódulos e módulos no módulo de nível superior é repetidamente executada até que o algoritmo completo requerido seja obtido.

Residência em Software

Programação BOTTOM-UP

- A abordagem bottom-up funciona com camadas de abstração.
- A principal aplicação da abordagem bottom-up é testar como cada módulo fundamental é testado pela primeira vez antes de mesclá-lo ao maior.
- O teste é realizado usando certas funções de baixo nível.

Residência em Software

Comparação entre TOP-DOWN e BOTTOM-UP

Base para comparação	Abordagem TOP-DOWN	Abordagem BOTTOM-UP
Básico	Quebra o enorme problema em subproblemas menores.	Resolve o problema fundamental de baixo nível e integra-os em um problema maior.
Processo	Submódulos são analisados individualmente.	Examine quais dados devem ser encapsulados e implica o conceito de ocultação de informações.
Comunicação	Não é necessário na abordagem top-down.	Precisa de uma quantidade específica de comunicação.
Redundância	Contém informações redundantes.	Redundância pode ser eliminada.
Linguagens de programação	Linguagens de programação orientadas a estrutura / processo (ou seja, C) seguem a abordagem top-don.	Linguagens de programação orientadas a objetos (como C ++, Java, etc.) seguem a abordagem bottom-up.

Residência em Software

Principais Diferenças

- Enquanto a abordagem **top-down** decompõe uma tarefa grande em subtarefas menores, a **bottom-up** escolhe resolver primeiro as diferentes partes fundamentais da tarefa diretamente e depois combinar essas partes em um programa.

- Na abordagem **top-down** os submódulos são processados separadamente, enquanto que na bottom-up é implementado o conceito de ocultação de informações e de dados encapsulados.

- A abordagem **top-down** não obriga a comunicação entre os módulos enquanto que a **bottom-up** necessita de interação entre os módulos fundamentais separados, de modo a que se possa posteriormente combiná-los.

Residência em Software

Principais Diferenças

- A abordagem **top-down** pode produzir redundância, enquanto a abordagem **bottom-up** não inclui informações redundantes.
- As **linguagens procedurais** de programação, como Pascal, COBOL e C, seguem uma abordagem preferencialmente top-down.
- Em contraste, **linguagens de programação orientadas a objetos**, como C ++, Java, C #, Perl, Python, seguem preferencialmente a abordagem bottom-up.

Residência em Software

Modularização - Coesão e Acoplamento

- Em modularização de software, **coesão** e **acoplamento** são dois conceitos importantes que descrevem a qualidade da organização e interdependência dos módulos em um sistema.
- Esses conceitos são fundamentais para criar sistemas bem estruturados, fáceis de entender, manter e evoluir ao longo do tempo.
- **Coesão** e **Acoplamento** são princípios originalmente introduzidos como parte do projeto estruturado.
- **Acoplamento** foca em aspectos de relacionamentos entre módulos, enquanto **Coesão** enfatiza a consistência interna de um módulo.

Acoplamento

Acoplamento é a **medida da força de associação entre módulos**.

Um alto acoplamento dificulta manutenções no sistema. **Módulos com alto acoplamento são mais difíceis de entender e manter.**

Alto Acoplamento **leva à propagação de efeitos colaterais** quando uma mudança é realizada no sistema.

Residência em Software

Em outras palavras...

- O acoplamento refere-se ao grau de dependência entre módulos em um sistema.
- Um sistema com baixo acoplamento possui módulos independentes, onde as alterações em um módulo têm poucas ou nenhuma repercussão em outros módulos.
- Por outro lado, um sistema com alto acoplamento significa que os módulos são altamente interdependentes, o que torna difícil fazer alterações em um módulo sem afetar outros.
- O objetivo é ter um acoplamento fraco entre módulos, para que cada módulo seja uma unidade independente e fácil de modificar ou substituir sem afetar outros módulos.

Residência em Software

Tipos de Acoplamento (em ordem crescente)

Acoplamento de Dados: módulos que se comunicam por parâmetros.

Acoplamento de Imagem: dois módulos são ligados por imagem se eles se referem à mesma estrutura de dados.

Acoplamento de Controle: um módulo passa para o outro um grupo de dados (flags) destinado a controlar a lógica interna do outro.

Residência em Software

Tipos de Acoplamento (em ordem crescente)

Acoplamento Comum: dois módulos possuem acoplamento comum se eles se referem à mesma área de dados.

Acoplamento de Conteúdo: dois módulos apresentam acoplamento de conteúdo se um faz referência ao interior do outro: por exemplo, se um módulo desvia a sequência de instruções para dentro do outro.

Coesão

É a medida da intensidade da associação funcional dos elementos em um módulo.

Residência em Software

Coesão

- Coesão refere-se ao grau de interdependência interna dos elementos (funções, classes, ou módulos) dentro de um módulo.
- Um módulo coeso realiza uma única tarefa ou possui uma única responsabilidade bem definida.
- Quanto mais coeso um módulo é, mais concentrada é a sua funcionalidade e menor é a sua complexidade interna.
- Isso torna o módulo mais fácil de entender, testar e modificar, uma vez que suas funcionalidades estão intimamente relacionadas.
- O ideal é alcançar uma coesão funcional alta, ou seja, módulos com uma única responsabilidade bem definida.

Residência em Software

Tipos de Coesão

- **Coesão Funcional**:
  - O módulo executa uma única tarefa ou funcionalidade.
- **Coesão Sequencial**:
  - As tarefas são executadas em sequência, uma após a outra.
- **Coesão Comunicacional**:
  - As tarefas estão relacionadas ao mesmo conjunto de dados.
- **Coesão Temporal**:
  - As tarefas devem ser executadas no mesmo período de tempo.
- **Coesão Procedural**:
  - As tarefas estão agrupadas arbitrariamente.

Residência em Software

Coesão Funcional

- Contém elementos que contribuem para a execução de uma e apenas uma tarefa relacionada ao problema.
- **Exemplo**:
  - Calcular Cosseno
  - Verificar Sintaxe no Texto
  - Calcular ponto de impacto de um míssil

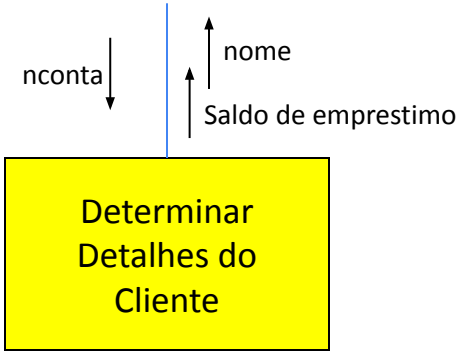
Residência em Software

Coesão Sequencial

- Elementos estão envolvidos em atividades onde os dados de saída de uma atividade servem como dados de entrada para a próxima.
- **Exemplo**:
  - Formatar e imprimir relatório Cliente

Comunicacional

- Os elementos contribuem para atividades que usem a mesma entrada ou a mesma saída.
- **Exemplo**:



Residência em Software

Procedimental ou Procedural

- Elementos estão envolvidos em atividades diferentes e possivelmente não relacionadas, nas quais flui controle de uma atividade para a outra:

1. Limpar utensílios da refeição anterior
2. Preparar peru para assar
3. Fazer telefonema
4. Tomar banho
5. Lavar vegetais
6. Pôr a mesa

Tendem a ser compostos por algumas funções pouco relacionadas entre si (exceto por serem executadas numa certa ordem numa certa hora).

Típico: dados recebidos e devolvidos tem pouca relação. Costumam devolver resultados parciais, flags e chaves.

Apelido: Faz pela metade

Residência em Software

Coesão Temporal

- Elementos estão envolvidos em atividades unicamente relacionadas no tempo.
  1. Colocar garrafas de leite para fora
  2. Colocar o gato para fora
  3. Desligar a tv
  4. Escovar os dentes

- **Exemplo**:

  Inicializar_Variaveis, Fechar_Arquivos e etc.

Residência em Software

Coesão Lógica

- Os elementos contribuem para atividades da mesma categoria geral, com as atividade a serem executadas sendo selecionadas for a do módulo.
- **Exemplo**

  Imprimir_Relatórios

Residência em Software

Coesão Coincidental

- Os elementos estão juntos por acaso.
- Fazem uma miscelânea de funções diferentes.

Arvore de decisão