Universidade Federal do Espírito Santo
Centro Norte do Espírito Santo - CEUNES

Linguagens de Programação

Prof. Francisco de Assis S. Santos, Dr.

2021/2

Apresentação da Disciplina

Conteúdo:

Propriedades desejáveis numa ling de prog. Amarrações (vinculações). Valores e tipos de dados. Variáveis e constantes. Expressões e comandos. Modularização. Polimorfismo. Exceções. Estudo comparativo de linguagens. Programação Lógica.

Pré-requisitos:
Estrutura de Dados I
Linguagem de Programação Orientada a Objetos

Bibliografia básica:

VAREJÃO, F.M. Linguagens de Programação: conceitos e técnicas. Rio de Janeiro: Campus, 2004.
SEBESTA, R. W. Conceitos de Linguagens de Programação. 5.ed. São Paulo: Bookman, 2003.
WATT, D. A. Programming Languages: concepts and paradigms. Prentice-Hall.

Razões para Estudar LPs

- Maior capacidade de desenvolver soluções computacionais para problemas
- Maior habilidade ao usar uma LP
- Maior capacidade para escolher LPs apropriadas
- Maior habilidade para aprender novas LPs
- Maior habilidade para projetar novas LPs

Papel das LPs nos PDS

- O objetivo de LPs é tornar mais efetivo o Processo de Desenvolvimento de Software (PDS)

- PDS visa geração e manutenção de software de modo produtivo e garantia de padrões de qualidade

Papel das Lps nos PDS

- Principais Propriedades Desejadas em um Software

  - Confiabilidade
  - Manutenibilidade
  - Eficiência

Papel das Lps nos PDS

- **Etapas do PDS**

  - **Especificação de Requisitos**: estudo de viabilidade
  - **Projeto do Software**: linguagem mais adequada ao método de projeto
  - **Implementação**: ferramentas
  - **Validação/Testes**: depuradores, testes caixa branca e caixa preta.
  - **Implantação:** Disponibilizar aos usuários
  - **Manutenção**: modularização, escalabilidade

Propriedades de Linguagens de Programação

Propriedades desejáveis em LPs

- **Legibilidade**
  - Identação errada

```
if (x>1)
  if (x==2)
x=3;
else
x=4;
else
x=5;
```

Propriedades desejáveis em LPs

- **Legibilidade**
  - Identação certa

```
if (x>1)
     if (x==2)
       x=3;
     else
       x=4;
else
   x=5;
```

Propriedades desejáveis em LPs

- **Legibilidade**
  - `else flutuante`

```
if (x>1)
   if (x==2)
      x=3;
else
      x=4;
```

  - Marcadores de Blocos

```
BEGIN ... END          (Pascal)
{ ... }                (C)
begin ... end-loop     (Ada)
```

Propriedades desejáveis em LPs

- **Legibilidade**
  - **`Identificadores com o mesmo nome de palavras reservadas`**

    `DO , END, INTEGER, REAL` (FORTRAN)

  - Desvios Incondicionais

    `goto`

Propriedades desejáveis em LPs

- **Confiabilidade**
  - **Declaração de Tipos**

```
// exemplo de baixa confiabilidade
boolean u = true;
int v = 0;
while (u && v < 9) {
     v = u + 2;   // deveria ser v = v  +2;

     if   (v == 6)  u = false;
}
```

Propriedades desejáveis em LPs

- **Confiabilidade**
  - **Tratamento de Exceções**

```
// exemplo de boa confiabilidade
    try {
          System.out.println(a[i]);
    } catch (IndexOutofBoundsException) {
                System.out.println("Erro  de
      Indexação");
    }
```

Propriedades desejáveis em LPs

- **Facilidade de Aprendizado**
  - **Excesso de Características é Prejudicial**

```
c = c + 1;
c+=1;
c++;
++c;
```

- **Modificabilidade**

```
- const float VALOR_MAXIMO = 30.5;
```

Propriedades desejáveis em LPs

- **Reusabilidade**

```
void troca(int *x, int *y) {
  int z = *x;
  *x = *y;
  *y = z;
}
```

- **Portabilidade**
  - Rigor no Projeto
  - Pode Contrastar com Eficiência

Exercícios

1) Desenvolva um bloco de código em C/C++/Java para calcular a área de três tipos de figuras geométricas: Triângulo, Circulo e Quadrado. Utilizar uma estrutura chamada figura geométrica que contenha os parâmetros dos tipos de figuras geométricas e posteriormente construir funções/métodos para o cálculo de cada tipo específico de figura.

O bloco de código deve aplicar as propriedades de Legibilidade, Confiabilidade, Facilidade de aprendizado, Reusabilidade e Modificabilidade.

Amarrações (Vinculações)

Conceituação

- Amarração é uma associação entre entidades de programação, tais como entre uma variável e seu valor ou entre um identificador e um tipo

- Enfoque na amarração de identificadores a entidades de programação (constantes, variáveis, procedimentos, funções e tipos)

Identificadores

- Identificadores são cadeias de caracteres definidas pelos programadores para servirem de referência a entidades de computação
- Objetivam aumentar a legibilidade, redigibilidade e modificabilidade
- LPs podem ser *case sensitive* e limitar o número máximo de caracteres
- Alguns identificadores podem ter significado especial para a LP
  - Palavras reservadas: if, int em C
  - Palavras Pré-definidas: printf, fopen em C

Tempos de Amarração

- Amarrações Estáticas:

Identificador ou Símbolo	Entidade	Tempo de Amarração
*	•Operação de multiplicação	•Projeto da LP
int	•Intervalo de inteiros	•Projeto da LP (JAVA) •Implementação do compilador(C)
variável	•Tipo	•Compilação (C) •Execução (polimorfismo em C++)
função	•Código correspondente da função	•Carga do programa
variável	•Área de memória	•Carga do programa

Tempos de Amarração

- Amarração Dinâmica:

Identifica-dor ou Símbolo	Entidade	Tempo de Amarração
Variável local	•Área de memória •Valor	•Execução

Ambientes de Amarração

- A interpretação de comandos e expressões, tais como a = 5 ou g(a + 1), dependem do que denotam os identificadores utilizados nesses comandos e expressões
- Um ambiente (ou *environment*) é um conjunto de amarrações
- Cada amarração possui um determinado escopo, isto é, a região do programa onde a entidade é visível

Ambientes de Amarração

- Amarração de identificador a duas entidades distintas no mesmo ambiente

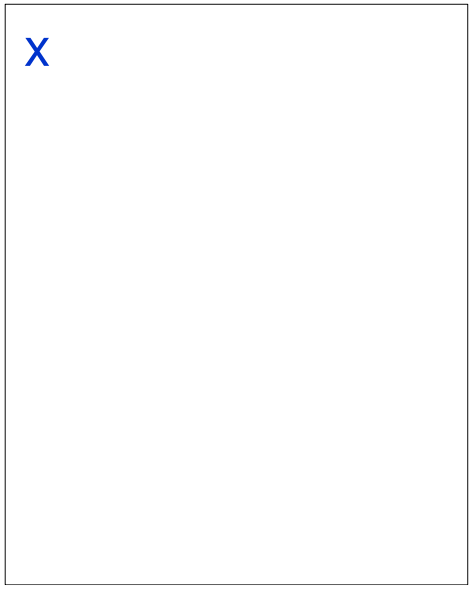
Em C++:

```
int a = 13;
void f() {
     int b = a;
     int a = 2;
     b = b + a;
}
```

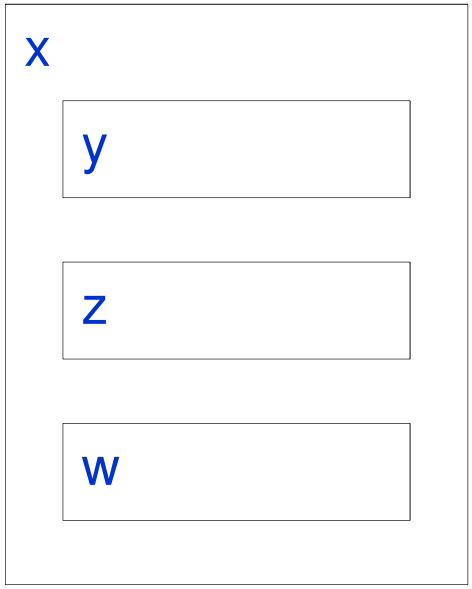
Escopo

- Estático
- Dinâmico

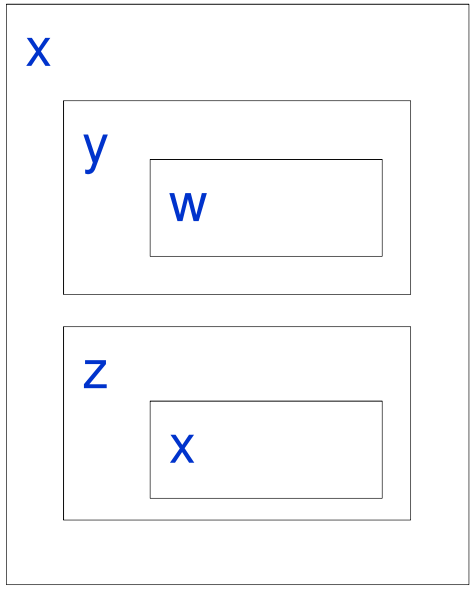
Escopo Estático



Bloco Monolítico



Blocos Não Aninhados



Blocos Aninhados

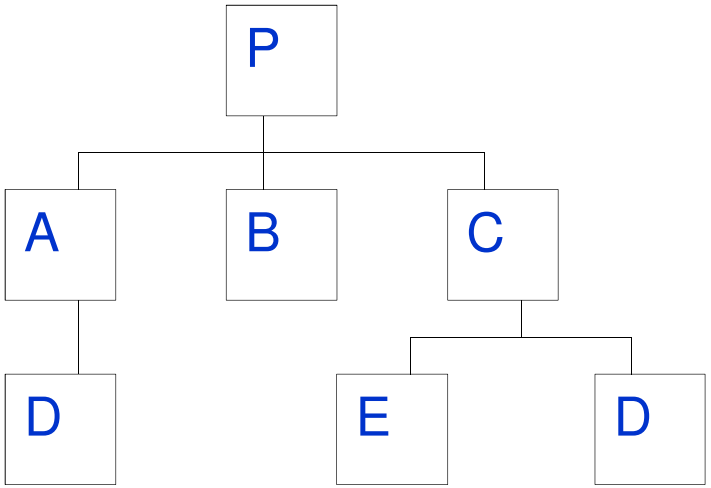
Escopo Estático

- Ocultamento de entidade em blocos aninhados: ficam inacessíveis

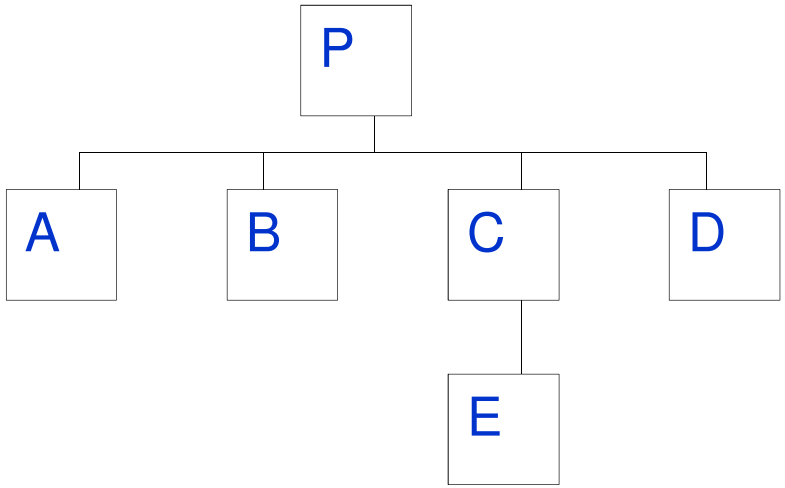
```
void main() {
    int i = 0,  x = 10;
    while (i++ < 100) {
      float x = 3.231;   // como acessar o x com valor 10?
      printf("x = %f\n", x*i);
    }
}
```

Escopo Estático

- Problemas com Estrutura Aninhada
  Como fazer D ser visível por A e C ?



bloco D repetido.
Visível por A, C e E



bloco D, agora visível por P e B ?

Escopo Estático

- Estrutura de blocos de C adota abordagem mista: funções não são aninhadas e blocos internos às funções podem ser aninhados

```
int x = 10;
int y = 15;
void f() {
        if (y >x) {
            int z = x + y;
        }
}
void g() {
    int w;
    w = x;
}
```

```
void main() {
    f();
     x = x + 3;
     g();
}
```

Escopo Dinâmico

```
procedimento sub(){
        inteiro x = 1;
         procedimento sub1(){
           escreva( x);
         }
         procedimento sub2(){
           inteiro x = 3;
         sub1();
         }
         sub2();
         sub1();
}
```

Escopo Dinâmico

- Problemas
  - Perda de eficiência: na verificação de tipos da variável e em seu acesso
  - Redução de legibilidade
  - Redução na confiabilidade
- Normalmente não usado por LPs
- Alguns exemplos de LPs: APL, SNOBOL4, LISP (versões iniciais) e PERL

Escopo (resumo)

- Estático
  - **definição do subprograma**
  - **tempo de compilação**
  - **texto do programa**
- Dinâmico
  - **chamada do subprograma**
  - **tempo de execução**
  - **fluxo de controle do programa**

Exercícios

2) Desenvolva um bloco de código em C/C++/Java para realizar a média aritmética de notas de N alunos. Na construção deste bloco utilize amarração de identificador a duas entidades distintas no mesmo ambiente. Também aplique o conceito de blocos aninhados no escopo estático.

Definições e Declarações

Definições e Declarações

- **Definições** produzem amarrações entre identificadores e entidades criadas na própria definição, ou seja, que fazem uso de uma memória nova: int i;

- **Declarações** produzem amarrações entre identificadores e entidades já criadas ou que ainda o serão: i = 5;

- Localização de definições de variáveis em C++

```
void f() {
   int a = 1;
   a = a + 3;
   int b = 0;  // compiladores mais antigos do C não aceitam
   b = b + a;
}
```

Declaração de Constantes

- Em C e C++

```
const float pi = 3.14;

#define pi 3.14   // substitui em tempo de pré-compilação
```

- Em JAVA

```
final float pi  = 3.14;
```

Definições e Declarações de Tipos

Definições Tipos em C: Palavras-Chave

```
struct data {
  int d, m, a;
};
```

```
union angulo {
 int graus;  //4 bytes
 float rad;  //4 bytes
             //Total = 8 bytes???
};
```

```
enum dia_util {
    seg,  ter, qua,
          qui, sex
};
```

Declarações Tipos em C

```
union angulo curvatura;
typedef struct data aniversario;
```

Definições e Declarações de Tipos

Definições Tipos em C: ENUM

```
#include <stdio.h>
#include <conio.h>
  enum meses_do_ano {Janeiro = 1, Fevereiro, Marco, Abril,   Maio, Junho, Julho, Agosto, Setembro,
Outubro, Novembro,  Dezembro} meses;

int main(void) {
 printf("Digite o numero do mes: ");
scanf("%d",&meses); //Testando se o valor está na faixa válida usando os valores da enum
if((meses >= Janeiro) && (meses <= Dezembro))
  {  switch(meses){
      case Janeiro: printf("%d - Janeiro",meses); break;
      case Fevereiro: printf("%d - Fevereiro",meses); break;
      case Marco: printf("%d - Marco",meses); break;
      //...até Dezembro
       }
     }
     else //senão estiver na faixa válida exibe mensagem
       printf("Valor INVALIDO!!!\n");
     printf("Os valores validos para os meses do ano sao: \n\n"); //Loop que exibe a faixa de valores válida
//Note que os valores da enum são na realidade inteiros //então podemos incrementa-los e usar no loop
   for(meses = Janeiro; meses <= Dezembro; meses++)
       printf("Mes: %d \n",meses);
   }
```

Exercícios

3) Uma loja de vestuário feminino precisa estabelecer códigos numéricos e sequenciais simples (Por exemplo, 101, 102, 103 ...) para: as cores, estações (primavera, primavera/verão, verão, alto verão, outono/inverno e outono e inverno) e tamanhos de suas roupas (pequeno, médio, médio-grande, grande, extra-grande).

Faça três funções/métodos em java/C/C++ aplicando definição e declaração de enum que permitam identificar e imprimir essas codificações.

Definições de Variáveis

- Definições de Variáveis em C

```
int k;
union angulo ang;
struct data d;
int *p, i, j, k, v[10];
```

- Definições com Inicialização

```
int i = 0;
char virgula = ',';
float f, g = 3.59;
int j, k, l = 0, m=23;
```

Definições de Variáveis

- Definições com Inicialização Dinâmica

```
void f(int x) {
   int i;
   int j = 3;
   i = x + 2;
   int k = i * j * x;
}
```

- Definições com Inicialização em Variáveis Compostas

```
int v[3] = { 1, 2, 3 };
```

Definições e Declarações de Variáveis

- Declaração de Variáveis em C

```
extern int a;  // compilador não gera espaço para ela
               // pois já é gerado por um módulo externo
```

- Declaração de Variáveis em C++

```
int r = 10;      //  definição de 'r'
int &j = r;      //   declaração de 'j' - referência para 'r'
j++;             // altera implicitamente 'r' para 11
```

Definições e Declarações de Subprogramas

- Definição de Subprogramas em C

```
int soma (int a, int b) {
 return a + b;
}
```

- Declaração de Subprogramas em C

```
int incr (int);  // o protótipo usado na Ling C é uma declaração de função

void f(void) {
   int k = incr(10);
}

int incr (int x) {   // aqui a função está sendo definida
   x++;
   return x;
}
```

Definições Compostas Recursivas

- Definição Recursiva de Função em C

```
float potencia (float x, int n) {
   if (n == 0) {
      return 1.0;
    } else if (n < 0) {
      return 1.0/ potencia (x, -n);
    } else {
      return x * potencia (x, n - 1);
    }
}
```

- Tipo Recursivo em C

```
struct lista {  int
   elemento;
   struct lista * proximo;
};
```

Definições e Declarações de Variáveis

- Declaração de Variáveis em C

  register int a;

  Variáveis como register acessam o acumulador diretamente no processador;

  Os registradores possuem pouco espaço para armazenamento. No entanto, estão no topo das memórias de alta velocidade;

  Declarar uma variável do tipo register é muito útil para laços significativos;

  Variáveis como register tratam-se de um pedido ao processador que pode ou não ser atendido.

Definições e Declarações de Variáveis

- Declaração de Variáveis em C

register int a; // Não deve ser uma variável global

```
int main (void){
register int count;  /*  Mas como comparar o tempo de execução com e sem register? */
int TAM=100;
int *vetor;
vetor = (int*) calloc (TAM, sizeof(int));
  if (vetor == NULL)
    printf ("** Erro: Memoria Insuficiente **");
 for (count=0;count<TAM;count++) {
     vetor[count]=count;
     printf (" %d ",vetor[count]);
 }
return 0;
}
```

Exercícios

4) Construir uma rotina em C/C++ para calcular o tempo de execução do seguinte processo computacional:

Utilizando, inicialmente, duas variáveis inteiras como register para produzir laços de repetições para linhas e colunas de uma matriz dinamicamente alocada, a qual receberá o produto dos índices das linhas e colunas. Repetir a execução 5 vezes e em seguida, retirar a definição de register e observar o tempo de execução por mais 5 vezes de execução.

Matri[l][c] = l*c;

OBS: Usar o comando clock() da biblioteca time.h;