

INFO1120 – TCP – Slides – Arquivo 5

# ***Técnicas de Construção de Programas***

**Prof. Marcelo Soares Pimenta**

*mpimenta@inf.ufrgs.br*

Porto Alegre, agosto a dezembro de 2011

# Leitura Recomendada

---

- “Problemas Gerais no Uso de Variáveis”, Cap. 10  
(original cap 10)

do livro McConnell, Steve *Code Complete – Um Guia Prático para Construção de Software*, 2ª edição, 2005.

PDF do original em inglês disponíveis no moodle da disciplina

# Convenções (standards) de programação

---

- Uso de variáveis
- Identificadores significativos
- Tipos de Dados
- Codificação

# Uso de tipos de dados básicos

---

- Números em Geral
  - Evite números literais
    - Ex: PI, TETO-SALARIAL, etc
    - Confiabilidade e facilidade de mudanças, legibilidade
    - Únicos tolerados: 0 e 1
      - For i:=0 to TOTALEMP ..... ;    total := total + 1
  - Explícite conversões de tipo e evite comparações misturadas (ver **WARNINGS** do compilador)
  - Antecipe divisões por zero

# Uso de tipos de dados básicos

---

- Inteiros

- Atenção à DIVISÃO INTEIRA  $7 \text{ DIV } 10 \nlessgtr 0.7$
- Atenção a overflow (MAXINT sem sinal, com sinal, etc)
  - 8 bits: c/ sinal: -128 a 127 , s/sinal: 0 a 255
  - 16 bits: c/sinal: -32768 a 32767 , s/sinal: 0 a 65535
  - (signed) int i:= 250 \* 300 = 9465 (!!!) - 75000-65535
  - Inclusive para valores intermediarios:  $i * i * i / 10000$ ;
  - Na dúvida, use inteiro longo ou float

# Uso de tipos de dados básicos

---

- Ponto flutuante
  - Problemas graves decorrentes da **precisão**, então:
    - Evite adições e subtrações com números de grandezas muito diferentes
      - P.ex:  $1.000.000 + 0.1$  às vezes resulta 1.000.000
      - Dica: Ordene e some a partir dos menores valores -> reduz problemas de arredondamento
    - Evite comparações de igualdade
      - $0.1 * 10$  nem sempre é igual a 1.0
      - Dica: função de comparação com um delta de tolerância
    - Antecipe erros de arredondamento
      - Use var de precisao dupla (double)
      - Ache soluções alternativas – BCD: decimais codif. em binários
    - Veja suporte da linguagem a dados específicos e use-o:
      - P.ex. Currency

# Uso de tipos de dados básicos

---

- Chars e Strings
  - Evite literais no código
    - Ex: ESCAPE, NOVA-LINHA, MSG-ERRO-34
    - Confiabilidade e facilidade de mudanças, legibilidade
  - Em caso de internacionalização, use Unicode (conj. de chars internacionais)
  - Torne bem localizadas as definições
  - Torne bem localizadas as conversões explícitas
  - Strings em C:
    - Cuidado especial:
      - Manipulação via strcmp(), strcpy(), strlen() e não ==, =, etc

# Uso de tipos de dados básicos

---

- Booleanos

- Uso para tornar código mais claro, simples e documentado

terminou := (indice > NUM-ELEM);

achou := (vetor[indice] == chave-busca);

If (achou || terminou) { ...

- Se linguagem não tem, crie seu tipo Booleano

- Ex. para C

Typedef int BOOLEAN

ou

Enum Boolean {

True = 1;

False = ! (True)

};



# Uso de tipos de dados básicos

---

- Constantes nomeadas (1/2)

- Use-as consistentemente
  - Não duplique definições , centralize-as
- Use-as em declarações e manipulações

```
CONST MAX-ALUNOS-TURMA = 40
```

```
...
```

```
Type Disciplina {  
  codigo : integer;  
  lista-alunos: array [1..MAX-ALUNOS-TURMA] of string;  
  ....
```

```
for indice:=1 to MAX-ALUNOS-TURMA do ....
```

- Se linguagem não as suporta, crie-as com variáveis ou classes

# Uso de tipos de dados básicos

---

- Constantes nomeadas (2/2)
  - Evite literais, mesmo os “confiáveis”

O que é mais claro?

```
for i:=1 to 12
```

```
    lucro [i] := receita[i] – despesa[i];
```

OU

```
for mes:= JANEIRO to DEZEMBRO
```

```
    lucro [mes] := receita[mes] – despesa[mes];
```

# Uso de tipos de dados básicos

---

- Arrays

- Atenção aos limites (inf e sup) do array !!
  - Há linguagens que não os verificam ...
  - Dica 1: no teste verifique 1º elem e seu sucessor, último elem e seu antecessor;
  - Dica 2: use constantes para definir e referenciar limites dos arrays !!
- Atenção ao cruzamento de atribuições:  
elemento[i] := elemento [j] + peso;
- Atenção à ordem dos arrays multidimensionais:  
matriz-A [i,j] := matriz-B[j,i] – matriz-c [i, j]

INFO1120 – TCP

# ***Técnicas de Construção de Programas***

**Prof. Marcelo Soares Pimenta**

*mpimenta@inf.ufrgs.br*

**Slides – Arquivo 7**

# Convenções (standards) de programação

---

- Uso de variáveis
- Identificadores significativos
- Tipos de Dados
- Codificação

# Uso de tipos de dados incomuns

---

- Estruturas

- Agrupar explicitamente informações relacionadas e facilitar trabalho de manutenção

- Ex: nome, endereço, fone, escolaridade, sexo, etc

**Empregado** { nome, endereço, fone, escolaridade, sexo }

- Simplificar operações sobre blocos de dados

- Ex. 

nome := oldnome;	empregado:=oldempregado;
fone := oldfone;	
....etc	

- Simplificar listas de parâmetros

- Ex. Rotina (nome,endereço, fone, escolaridade, sexo)

vs

Rotina (empregado)

# Uso de tipos de dados incomuns

---

- Ponteiros
  - Localize suas operações de ponteiro em rotinas ou classes, minimizando os locais onde são acessados;
  - Exclua ponteiros no mesmo escopo onde foram alocados e na ordem de alocação (especialmente em listas encadeadas)
  - Para clareza, simplifique expressões de ponteiro complicadas e/ou use ponteiros extra – evite indireção de indireção
  - Ponteiros em C:
    - Ponteiros (\*) <> referências (&)
    - Use sizeof() para determinar tamanho de var alocada

# Uso de tipos de dados incomuns

---

- Globais
  - Problemas freqüentes :
    - Alterações involuntárias de valor (efeito colateral)
    - Compartilhamento por código reentrante (diferentes cópias do mesmo programa)
    - Dificulta reuso de código
    - Globais prejudicam modularidade
  - Quando usar globais?
    - Preservação de valores globais (status do programa, p.ex)
    - Simulação de constantes nomeadas (quando linguagem NÃO possuir)
    - Eliminar dados “itinerantes” (evitar rotinas “intermediárias”)
  - USO DISCIPLINADO:
    - Rotinas de acesso para centralizar controle e proteger de alterações indevidas



# Convenções (standards) de programação

---

- Uso de variáveis
- Identificadores significativos
- Tipos de Dados
- Codificação

# Dicas de Codificação

---

- Código Linear
- Condicionais
- Loops
- Controles incomuns
- Manipulação de tabela
- Questões gerais de controle

# Código Linear

---

- Código que deve seguir uma ordem específica:
  - Torne evidentes as dependências
    - Nomes de rotinas
    - Parâmetros
  - Documente dependências ocultas com comentários
- Código cuja ordem não importa
  - Organizar o código para ser lido de cima para baixo, evitando saltos
    - Referências centralizadas a mesmos dados
  - Agrupar instruções relacionadas

# Condicionais (1/2)

---

- Uso de IFs
  - Escreva primeiro o caso “normal” (no IF), depois as exceções (no ELSE)
  - Atenção aos valores-limite  $<$ ,  $>=$ ,  $<=$ ,  $>$
  - Simplifique testes complicados com funções booleanas
    - Ex. If ( IsDigit (inputChar) ) ...
  - Em IFs encadeados, teste primeiros os casos MAIS comuns
  - Certifique-se de que TODOS casos estão cobertos
  - Substitua IFs encadeados por outros comandos (Case)

# Condicionais (2/2)

---

- Uso de CASEs
  - Classificação mais eficiente dos casos
    - Caso “mais” normal primeiro
    - Ordenados por frequência
    - Ordenados lexicograficamente (alfabética, etc)
  - Mantenha as ações simples
    - Código curto associado às cláusulas do Case
  - Cláusula DEFAULT quando é legítima ou erro

# Loops(1/3)

---

- Seleccionando o tipo de loop
  - Contado : num determinado de vezes
  - Avaliado continuamente
  - Infinito - sist. Embarcados (marca-passo, microonda, celular,etc)
    - Forma padrão: `while (true) { ..... }`
  - Com teste no início (while) ou no fim (repeat), incrementos constantes ou não
    - Não use FOR ao invés do WHILE, nem WHILE ao invés do FOR
- Controlando o loop
  - Simplifique o num de fatores que afeta o loop
    - Atribuição inicial, aninhamento incorreto, término incorreto, incremento errado ou ausente, indexação errada de elemento, etc
  - Trate o interior do loop como uma rotina – controle fica fora !!
    - Corpo do loop é caixa preta
    - Evite loops vazios (expressão como atribuição e corpo vazio !!!)

# Loops(2/3)

- Entrando no loop
  - A partir de um único local;
  - Com atribuição de variáveis de controle logo antes do loop
- Meio do loop
  - Controles no início ou fim do corpo, nomes significativos
  - Cada loop executando apenas uma função - ~rotina
- Fim do loop
  - Certifique-se de que termina e torne evidentes suas condições de término;
  - Não manipule índice do FOR para forçar término
    - Ex. `for (int i = 0; i < 100; i++ ) {`

```
....  
    if (....) {  
i = 100;  
    }  
}
```

# Loops(3/3)

## – Fim do loop

- Evite código que dependa do contador do loop
  - Use outras variáveis e deixe o contador como local
- Saindo antecipadamente de loops
  - Use instruções **break** (e **continue**) mas não espalhadas;
- Verificando limites
  - Loops têm 3 ocorrências de interesse a verificar:
    - » Primeira iteração
    - » Iteração intermediária (arbitrária)
    - » Última iteração
  - Se houver casos especiais diferentes destas, verifique-os também
- De preferência, limite as variáveis de índice ao próprio loop

## – Comprimento ideal do loop

- Curtos para serem vistos de uma “olhada”- uma página
  - Mova partes para rotinas
- Limite aninhamento – máx 3 níveis



# Controles incomuns(1/2)

---

- Múltiplos retornos de uma rotina
  - Use `return` (`exit`, `quit`, etc) para aumentar legibilidade, mas minimize número de retornosw em cada rotina
  - Programação estruturada: “*single entry, single exit*”
- Recursividade
  - Certifique-se de que ela termina (pode usar var de segurança)
  - Restrita a uma única rotina
  - Examine alternativas à recursividade, antes de usá-la
    - Cálculo recursivo de Fatorial e Fibonacci é elegante mas complexa e ineficiente

# Controles incomuns(2/2)

---

- Desvio incondicional (*go to*) – Questão Dogmática

- Se for usar, use adequadamente: para frente !!!
- Exercício sobre re-escrita de código sem *go to* :

```
if (statusOk) {  
    if ( dataAvailable) {  
        importantVariable = x;  
        goto MID-LOOP;  
    }  
}  
else {  
    importantVariable = GetValue();  
MID-LOOP:  
    // muito código.....
```

- Alternativas equivalentes com ou sem uso de novas rotinas !!!
- Ver CACM de março a dezembro de 1987:
  - 17 re-escritas sem *go to* equivalentes de um algoritmo “superior com *go to*”(7 linhas)

# Manipulação de tabela

---

- Problemas típicos:
  - Como pesquisar na tabela ?
    - Acesso Direto aos elementos
      - Ex. Tabela de dados NumDiasCadaMês, indexada pelo mês
    - Acesso Indexado (~ hash)
      - Tabela de índice + Tabela de Dados
    - Acesso tipo escada, para intervalos complicados
  - O que deve ser armazenado na tabela?
    - Dados
    - Ações ou ponteiros para ações (complicado)

# Questões gerais de controle(1/4)

---

- Expressões booleanas
    - Interferem em todos os caminhos não sequenciais de um código
    - Simplifique expressões, use parênteses e use funções booleanas
      - Transformações de DeMorgan
        - if (!displayOK || ! printerOK) ... equivale a
        - if (! ( displayOK && printerOK) ) ...
    - Atenção à forma de avaliação de cada linguagem
      - Cuidado com “lazy evaluation” (C++ , Java)
        - Ex1. if (algoFalso && algumaCondição)
        - Ex2. if (algoVerdadeiro || algumaCondição)
        - Ex3. if ( (denom != 0 ) && ( item / denom) > MIN-VALOR ) )...
- Divisão por zero NUNCA ocorre se denom == 0**

# Questões gerais de controle(2/4)

---

- Comparações com ZERO
  - Booleanas , implicitamente
    - Ex. while (! done)
  - Números com 0
    - Ex. while (balance != 0) **e não** while (! balance)
  - Em C, compare chars com terminador nulo ‘\0’
    - Ex. while (\*charPtr != TERMINADOR) **e não** while (\*charPtr)
  - Compare ponteiros com NULL
    - Ex. while (bufferPtr != NULL) **e não** while (bufferPtr)

# Questões gerais de controle(3/4)

---

- Instruções compostas (blocos)
  - Escreva pares de INICIO e FIM { }, juntos
  - Use INICIO e FIM para esclarecer condicionais
  - Evite aninhamentos (indentação) profundos de blocos

- Instruções NULAS

- Ex.

```
while( recArray.Read(index++) != recArray.EmptyRec() )  
    ;
```

- Avalie se código fica mesmo mais claro com elas
  - Chame a atenção (linha única, comentários, etc)
  - Crie uma macro ou função para elas - DoNothing()

# Questões gerais de controle(4/4)

---

- Estas dicas todas seguem os princípios originais da Programação Estruturada
  - Single entry , single exit
  - Construções necessárias e suficientes para escrita de QUALQUER algoritmo
    - SEQUÊNCIA, conj. de instruções executadas em ordem
    - SELEÇÃO , conj. de instruções executadas seletivamente (condicionalmente)
    - ITERAÇÃO, conj. de instruções executadas repetidamente
- Estruturas de controle: são responsáveis pela complexidade global do programa
  - Bom uso de estruturas , diminui complexidade
  - Complexidade ciclomática de McCabe
    - Pontos de decisão de uma rotina

# Leitura Recomendada

---

- “Questões Gerais de Controle”, Cap. 19 (original cap 19)

do livro McConnell, Steve *Code Complete – Um Guia Prático para Construção de Software*, 2ª edição, 2005.

PDF do original em inglês disponíveis no moodle da disciplina