INSTITUTO DE INFORMÁTICA UFRGS

UFRGS

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

Convenções (standards) de programação

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

Uso de variáveis (1)

- Evite declarações implícitas
  - Declare todas variáveis; verifique seus nomes
- Inicialize as variáveis
  - Ao declará-la float mediaTurma = 0.0;
  - Próximo ao seu 1º uso
  - Constantes NÃO são variáveis final Java ou const C
    - Valores inalterados após primeira definição
- Escopo de var
  - Abrangência da var: agrupar usos da mesma var
  - Tempo de vida de var: entre 1ª e última referência a ela
  - Minimizar escopo:
    - Iniciar var antes de usá-las em loops
    - Manter uma var o mais local possível

Uso de variáveis (2)

- Persistência:
  - De um bloco a “forever” (persistência em BDs)
  - Presuma que os dados não são persistentes
  - Habitue-se a declarar e iniciar todos dados imediatamente ANTES de serem usados

- Vinculação
  - Tempo de escrita x Tempo de compilação x Tempo de Execução
    - corBarra = 0xFF ; // 0xFF é valor hexa da cor azul
    - Private static final int COR_AZUL = 0xFF;
      Private static final int COR_BARRA = COR_AZUL;
      ....
      corBarra = COR_BARRA;

    - corBarra = ReadTitleBarColor ( ) ;

  - Em geral, quanto MAIS cedo a vinculação , MENOR a flexibilidade e a complexidade !

Uso de variáveis (3)

- Uma var tem .....exatamente UM propósito

```
temp = Sqrt (b*b -  4*a*c) ;
root[0] =  (-b + temp) /  (2 * a);
root[1] =  (-b - temp) / (2 * a);
....
temp = root[0];
root[0] = root[1];
root[1] = temp;
```

```
delta = Sqrt (b*b -  4*a*c) ;
      root[0] =  (-b + delta) / ( 2 * a);
 root[1] =  (-b - delta) / (2 * a);

temp = root[0];
      root[0] = root[1];
      root[1] = temp;
```

- Evite var com significados (múltiplos) ocultos (Acoplamento híbrido)
  - P.ex:
    - contPag se > 0 , num de páginas impressas
      se < 0 , erro de impressão

- Todas var declaradas devem ser usadas
  - Use alertas de verificadores (lint)

Análise Estática Automatizada

- Analisadores estáticos são ferramentas (sw) para processamento (análise automática) do código fonte
- Basicamente, fazem um “parse” do programa e tentam descobrir condições potencialmente suspeitas (na nomenclatura OO, “bad smells”) e alertá-las para a equipe de teste
- MUITO efetiva como auxílio a inspeção , mas apenas a complementa , não a substitue

O que pode ser analisado automaticamente?

Fault class	Static analysis check
Data faults	Variables used before initialisation Variables declared but never used Variables assigned twice but never used between assignments Possible array bound violations Undeclared variables
Control faults	Unreachable code Unconditional branches into loops
Input/output faults	Variables output twice with no intervening assignment
Interface faults	Parameter type mismatches Parameter number mismatches Non-usage of the results of functions Uncalled functions and procedures
Storage management faults	Unassigned pointers Pointer arithmetic

Ex. de resultado do analisador LINT

```
138% more lint_ex.c

#include <stdio.h>
printarray (Anarray)
  int Anarray;
{
  printf("%d",Anarray);
}
main ()
{
  int Anarray[5]; int i; char c;
  printarray (Anarray, i, c);
  printarray (Anarray) ;
}

139% cc lint_ex.c
140% lint lint_ex.c

lint_ex.c(10): warning: c may be used before set
lint_ex.c(10): warning: i may be used before set
printarray: variable # of args. lint_ex.c(4) :: lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) ::
lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) ::
lint_ex.c(11)
printf returns value which is always ignored
```

Identificadores significativos (1)

- Identificador descreve total e precisamente uma entidade
  - Diga o que ela representa: este deve ser seu nome !!
  - Exercício:
    - Identificadores que descrevam:
      - Número de pessoas na equipe olímpica do Brasil
      - Total geral dos cheques gravados até a presente data
      - Velocidade de um trem bala
      - Data atual
      - Linhas por página
- Tamanhos dos nomes:
  - Curtos demais têm pouco significado
  - Longos demais têm problemas de manipulação (limitações de alguns sistemas de arquivos)
    - Tam ideal : bom senso, mas (Gorla&Benander&Benander 1990) : 10 a 16 chars

Identificadores significativos (2)

- Identificadores de tipos de dados específicos
  - Índices de loops simples
    - i, j, k
    - Se loops aninhados ou var usada fora do loop , escolher um melhor : evitar erros de troca de i por j
  - Variáveis de status

• if (flag)	if (dataReady)
• if (statusFlag & 0x0F)	if (charType & PRINTABLE_CHAR)
• if (printFlag == 16)	if (reportType == Annual_report)
• if (computerFlag == 0)	if (recalcNeed == TRUE)

  - Variáveis booleanas
    - Nomes booleanos típicos: pronto, erro, achou, etc
    - Nomes que indiquem valor V ou F: status? arqFonte? origem?
    - Nomes positivos: if not (naoAchou)

Identificadores para métodos (1)

- Descreva tudo que método faz, nome suficientemente longo
  - Evite nomes longos demais ou “quebre” o método
- Evite nomes vagos e inexpressivos
  - TratarEntrada ( ), ExecutarServiços( ), ProcessarEntrada ( ) ....
- Não diferencie métodos unicamente pelo número
  - CalculaTotal1 ( ), CalculaTotal2 ( ) : abstração ??
- Descrição de valor de retorno é bom nome de função
  - idCliente.Proximo ( ), Impressora.EstaPronta ( )

Convenção (Java e C++)

- Nomes de Classes: ClassName
- Definições de tipo, enumerados e typedef: TypeName
- Var locais: localVariable
- Parâmetros: routineParameters
- Métodos e rotinas: MethodNames ( )
- Constantes : CONSTANTE
- Macros: MACRO

Identificadores a serem evitados

- Abreviações que induzem ao erro
  - falso – Feira de Amostra e Lançamento de Software
- Nomes com significados semelhantes
  - recordNum e numRecord
- Nomes semelhantes e significados diferentes
  - clientRecs e clientReps
- Evite numerais nos nomes : arquivo 1, arquivo2, etc
- Evite palavras grafadas erroneamente: menasEnergia
- Evite múltiplos idiomas
- Evite nomes padrão de tipos e métodos: get, set, int, float
- Evite nomes contendo caracteres difíceis de ler:
  - GRANDTOTAL e 6RANDT0TAL

Pontos chave

- Bons nomes são chave para legibilidade do código
- Nomes devem ser o mais específicos possível
- É preciso que uma instituição adote uma convenção para nomes que seja adequada para sua realidade.
- Abreviaturas raramente são necessárias mas se preciso use-as de modo controlado

- Um código é bem mais vezes lido do que escrito.
  - Nomes mais fáceis de ler devem ser preferidos em detrimento de nomes mais fáceis de escrever