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

Linguagens de Programação

Parte 04

Prof. Francisco de Assis S. Santos, Dr.

2021/2

Modularização

Introdução: Programação em Bloco Monolítico

- **Inviabiliza grandes sistemas de programação**
  - Um único programador pois não há divisão do programa
  - Indução a erros por causa da visibilidade de variáveis e fluxo de controle irrestrito
  - Dificulta a reutilização de código
- **Eficiência de programação passa a ser gargalo**

Introdução: Processo de Resolução de Problemas Complexos

- **Uso de Dividir para Conquistar**
  - Resolução de vários problemas menos complexos
  - Aumenta as possibilidades de reutilização
- **Técnicas de Modularização objetivam Dividir para Conquistar**
  - Tornam mais fácil o entendimento do programa
  - Segmentam o programa
  - Encapsulam os dados – agrupam dados e processos logicamente relacionados

Introdução: Sistemas de grande porte

- **Características**
  - Grande número de entidades de computação e linhas de código
  - Equipe de programadores
  - Código distribuído em vários arquivos fonte
  - Conveniente não recompilar partes não alteradas do programa

Introdução: Sistemas de grande porte

- **Módulo**
  - Unidade que pode ser compilada separadamente
  - Propósito único
  - Interface apropriada com outros módulos
  - Reutilizáveis e Modificáveis
  - Pode conter um ou mais tipos, variáveis, constantes, funções, procedimentos
  - Deve identificar claramente seu objetivo e como o atinge (coesão)

Modularização: Abstração

Fundamental para a Modularização

- Seleção do que deve ser representado
- Possibilita o trabalho em níveis de implementação e uso
- Uso dimensionado na Computação

Modularização: Abstração

- **Exemplos de uso na computação**
  - Comandos do SO
  - Assemblers
  - LPs
  - Programa de Reservas

- **Modos**
  - LP é abstração sobre o hardware
  - LP oferece mecanismos para o programador criar suas abstrações - este fundamenta a Modularização

Modularização: Tipos de Abstração

- **Abstrações de Processos**
  - Abstrações sobre o fluxo de controle do programa
  - Suprogramas – funções da biblioteca padrão de C (*printf*)
- **Abstrações de Dados**
  - Abstrações sobre as estruturas de dados do programa
  - Tipos de Dados – tipos da biblioteca padrão de C (*FILE*)

Modularização: Abstração de Processos

Subprogramas

- Permitem segmentar o programa em vários blocos logicamente relacionados
- Servem para reusar trechos de código que operam sobre dados diferenciados
- Modularizações efetuadas com base no tamanho do código possuem baixa qualidade
- Propósito único e claro facilita legibilidade, depuração, manutenção e reutilização

Perspectivas do Usuário e do Implementador do Subprograma

- **Usuário (programador que usa o subprograma)**
  - Interessa o que o subprograma faz
  - Como usar é importante
  - Como faz é pouco importante ou não é importante
- **Implementador (programador que faz o subprograma)**
  - Importante é como o subprograma realiza a funcionalidade

Perspectivas do Usuário e do Implementador do Subprograma: Exemplo

```
int fatorial(int n) {
   if (n<2) {
      return 1;
   } else {
      return n * fatorial (n – 1);
   }
}
```

- **Usuário**
  - Função fatorial é mapeamento de n para n!
- **Implementador**
  - Uso de algoritmo recursivo

Perspectivas do Usuário e do Implementador do Subprograma: Exemplo

```
void ordena (int numeros[50]) {
  int j, k, aux ;
  for (k = 0; k < 50; k++) {
          for (j = 0; j < 49; j++) {
             if (numeros[j] < numeros[j+1]) {
                aux = numeros[j];
                numeros[j] = numeros[j+1];
                numeros[j+1] = aux;
             }
          }
  }
}
```

→Usuário Ordenação de vetor de inteiros

→Implementador Método da bolha

Parâmetros: Exemplo

```
// Quais problemas na falta de parâmetros?
   int altura, largura, comprimento;
  int volume () {
        return altura * largura * comprimento;
   }
  main() {
     int a1 = 1, l1 = 2, c1 = 3, a2 = 4, l2 = 5, c2 = 6;
      int v1, v2;
             altura = a1; largura = l1;   comprimento = c1;
              v1 = volume();
              altura = a2; largura = l2; comprimento = c2;
              v2 = volume();
              printf ("v1: %d\nv2: %d\n", v1, v2);
  }
```

Parâmetros: Exemplo

- **Ausência reduz**
  - **Redigibilidade**
    - Necessário incluir operações para atribuir os valores desejados às variáveis globais
  - **Legibilidade**
    - Na chamada de volume não existe qualquer menção à necessidade de uso dos valores das variáveis altura, largura e comprimento
  - **Confiabilidade**
    - Não exige que sejam atribuídos valores a todas as variáveis globais utilizadas em volume

Parâmetros: Exemplo

- **Resolvem esses problemas**

```
int volume (int altura, int largura, int  comprimento){
    return altura * largura * comprimento;
}

main() {
    int a1 = 1, l1 = 2, c1 = 3, a2 = 4, c2 = 5, l2 = 6;
    int v1, v2;
    v1 = volume(a1, l1, c1);
    v2 = volume(a2, l2, c2);
    printf ("v1: %d\nv2: %d\n", v1, v2);
}
```

Parâmetros Reais, Formais e Argumentos

- **Parâmetro formal**
  - Identificadores listados no cabeçalho do subprograma e usados no seu corpo
- **Parâmetro real**
  - Valores, identificadores ou expressões utilizados na chamada do subprograma
- **Argumento**
  - Valor passado do parâmetro real para o parâmetro formal

Parâmetros Reais, Formais e Argumentos: Exemplo 01

```
float area (float r) {
    return 3.1416 * r * r;
}

main() {
    float diametro, resultado;
    diametro = 2.8;
    resultado = area (diametro/2);
}
```

Valores Default de Parâmetros

- **Em C++ (C não aceita)**

```
int soma (int a[ ], int inicio = 0, int fim = 7, int incr  = 1){
 // definição de default somente da direita para a  esquerda
        int soma = 0, i;
        for (i = inicio; i < fim; i+=incr)
            soma+=a[i];
        return soma;
    }
  main() {
     int pontuacao[ ] = { 9, 4, 8, 9, 5, 6, 2};
     int ptotal, pQuaSab, pTerQui, pSegQuaSex;
     ptotal = soma(pontuacao);
     pQuaSab = soma(pontuacao, 3);
     pTerQui = soma(pontuacao, 2, 5);
     pSegQuaSex = soma(pontuacao, 1, 6, 2);
  }
```

Lista de Parâmetros Variável: Exemplo operador OR

```
#include <stdarg.h>
  int ou (int n, ...) { //construção do comando OR
     va_list valores;          // inicializa a lista de args
     int i, r = 0;
     va_start (valores, n);    // define para valores n args
     for (i = 0; i < n; i++)
        if (va_arg (valores, int))  // lê um arg da lista
         r = 1;
     va_end (valores); //desaloca a lista
     return r;
  }
```

- Oferece maior flexibilidade à LP
- Reduz a confiabilidade pois não é possível verificar os tipos dos parâmetros em tempo de compilação

Lista de Parâmetros Variável: Exemplo operador OR

```
main() {
    printf (“%d\n”, ou (1, 3 < 2));
    printf (“%d\n”, ou (2, 3 > 2, 7 > 5));
    printf (“%d\n”, ou (3, 1 != 1, 2 != 2, 3 != 3));
    printf (“%d\n”, ou (3, 1 != 1, 2 != 2, 3 == 3));
 }
```

Exercícios

1) Desenvolver uma rotina para obter o resultado da operação lógica “and”, em C/C++, independente da quantidade de parâmetros ou argumentos a serem utilizados. Desenvolva também para o operador lógico “XOR”.

2) Assumindo a séria Fibonacci com 8 termos: 1 1 2 3 5 8 13 21, desenvolva uma rotina, em C/C++, capaz de realizar o produto entre os termos entre si. Porém, de forma flexível quanto a quantidade e quais termos entre os utilizados da série Fibonacci.