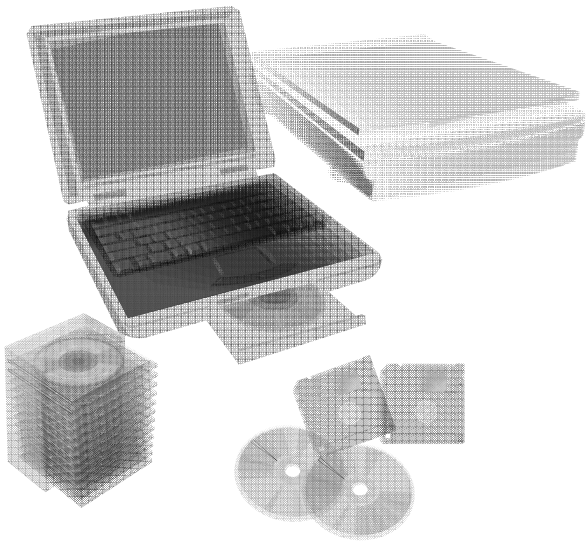
Introdução à Computação II



Documentação de programas

Profa.: ***Adriane Beatriz de Souza Serapião***

adriane@rc.unesp.br

Documentação de programas

- **Utilizada basicamente para:**
  - **depuração e análise de programas;**
  - **alterações de manutenção e evolução de programas;**
  - **alterações de adaptações (para outra aplicação ou outro equipamento).**
- **Há basicamente três tipos de documentação:**
  - **resumo do programa** (finalidades e objetivos do programa)**;**
  - **descrição detalhada do programa** (utilização)**; e**
  - **documentação interna** (o que cada comando ou grupo de comando faz)**.**
- **Os dois primeiros são chamados de documentação externa.**

Documentação de programas

- **Quanto à DOCUMENTAÇÃO INTERNA:**
  - **programas ou subprograma deve ter informação do tipo: nome do programa, autor, data (com as de revisões), finalidade, compilador (e versão) usado, etc.;**
  - **subprograma com: como chamar o subprograma, descrição dos argumentos, etc.;**
  - **nomes de variáveis: devem ser escolhidos apropriadamente para representar o melhor possível o seu significado no problema; podem ser comentadas para maior entendimento;**
  - **saídas: deve haver adequada documentação dos resultados de saída, facilitando a compreensão.**

Documentação de programas

- **Quanto a DOCUMENTAÇÃO EXTERNA, pode ser escrita na forma de relatório com:**
  - **identificação do programa, programador, datas, etc.;**
  - **finalidades e objetivos;**
  - **configuração da máquina necessária ("hardware");**
  - **linguagem de programação utilizada;**
  - **informações gerais e de operação;**
  - **subprogramas utilizados e como chamá-los;**
  - **restrições (se houver);**
  - **referências bibliográficas, com descrição da técnica utilizada (se houver);**
  - **Anexar algoritmo e uma listagem do programa fonte.**

Orientações gerais para comentar um programa

- **Não comentar é crime, comentar demais também. Algumas orientações gerais podem ser seguidas ao se introduzir comentários em um programa:**

1. **Colocar um cabeçalho no programa: Normalmente todo programa apresenta um cabeçalho contendo nome do programa, autores, data em que foi feito e objetivo do programa.**
2. **Não deixe de colocar comentários em seu programa.**
3. **Não comente demasiadamente seu programa: um programa que possui todas as linhas comentadas torna-se tão incompreensível quanto um programa sem comentários.**
4. **Comente apenas o necessário: Não comente linhas de comando que por si só já se explicam.**
5. **Comente declarações de variáveis: Ao comentar as declarações, indique qual o papel de cada variável no seu programa, como feito no programa anterior.**

Orientações gerais para comentar um programa

- **Não comentar é crime, comentar demais também. Algumas orientações gerais podem ser seguidas ao se introduzir comentários em um programa:**

6. **Comente funções e procedimentos: Assim como nas partes principais dos programas, é importante comentar *functions* e *procedures*, especificando em cabeçalhos próprios os objetivos de cada *procedure* e *function* e seus parâmetros de entrada e de saída (nomes e objetivos).**
7. **Comente fórmulas e expressões complicadas: Fórmulas e expressões tendem a ser complicadas de se entender, a menos que sejam explicadas.**
8. **Não complique, explique: Existem comentários que conseguem ser piores que o código original.**
9. **Comente o que realmente acontece: Nunca comente algum trecho de programa sem antes saber exatamente o que ele faz; você pode eventualmente dizer que o programa faz algo que na realidade ele não faz.**
10. **Use o bom senso.**

Indentação de programas

- **A sugestão quanto ao uso de espaços refere-se ao estilo de formatação típico da linguagem C, conhecido como *pretty-printing*.**

- **Esta regra é simples: Sempre que se precisar escrever uma instrução composta, indente-a em dois espaços à direita do resto da instrução corrente. Uma instrução composta dentro de outra instrução composta é indentada em quatro espaços, e assim por diante:**

```
if ...
{
  if ...
    instrução1;
  instrução2;
};
```


Indentação de programas

- **O *{* e o *}* de um mesmo bloco devem começar na mesma coluna. Assim, fica fácil descobrir qual *{* se refere a qual *}*.**

- **Todas as instruções que pertencem a um mesmo bloco devem estar indentadas em relação à coluna dos seus respectivos *{* e *}*. As duas regras acima se referem aos blocos *if...else* e *do...while* da mesma forma que ao bloco *{ ...}*.**

- **Tamanho usado para cada indentação: normalmente dois espaços em branco. Um único espaço não causa um efeito visual convincente, e muitos espaços podem eventualmente confundir quem está lendo o programa ou forçar quebra de linha. Use o bom senso.**

Indentação e comentários

- **À medida que os programas crescem, torna-se cada vez mais difícil ter controle sobre o que ele está fazendo. É fácil se perder tentando descobrir qual *{* se refere a qual *}*, qual *do* se refere a qual *while*, etc. Além disso, também fica complicado descobrir rapidamente qual parte do programa se refere a uma determinada ação. *Indentação* e *comentários* são dois passos que podem ser dados em direção a um maior entendimento do programa.**

- **Através do uso de indentações, consegue-se:**
  - **associar pares *{...}*, *case...break*, *if...else* etc.;**
  - **destacar, em estruturas de repetição, quais são as linhas que serão repetidas;**
  - **destacar, em estruturas de decisão, quais são as linhas relacionadas a cada possível decisão tomada;**
  - **destacar quais instruções pertencem a um mesmo bloco *begin...end*.**

- **O uso de comentários, por sua vez, torna possível:**
  - **esclarecer qual o papel de um determinado programa, procedimento, função ou trecho de programa;**
  - **esclarecer qual o papel das variáveis utilizadas no programa.**

Estilo de programação

Critérios de qualidade de um programa:

- **Integridade: refere-se à precisão das informações manipuladas pelo programa, ou seja, os resultados gerados pelo processamento do programa devem estar corretos, caso contrário o programa simplesmente não tem sentido.**

- **Clareza: refere-se à facilidade de leitura do programa. Se um programa for escrito com clareza, deverá ser possível a outro programador seguir a lógica do programa sem muito esforço, assim como o próprio autor do programa entendê-lo após ter estado um longo período afastado dele. O C favorece a escrita de programas com clareza e legibilidade.**

- **Simplicidade: a clareza e precisão de um programa são normalmente melhoradas tornando as coisa o mais simples possível, consistentes com os objetivos do programa. Muitas vezes torna-se necessário sacrificar alguma eficiência de processamento, de forma a manter a estrutura do programa mais simples.**

Estilo de programação

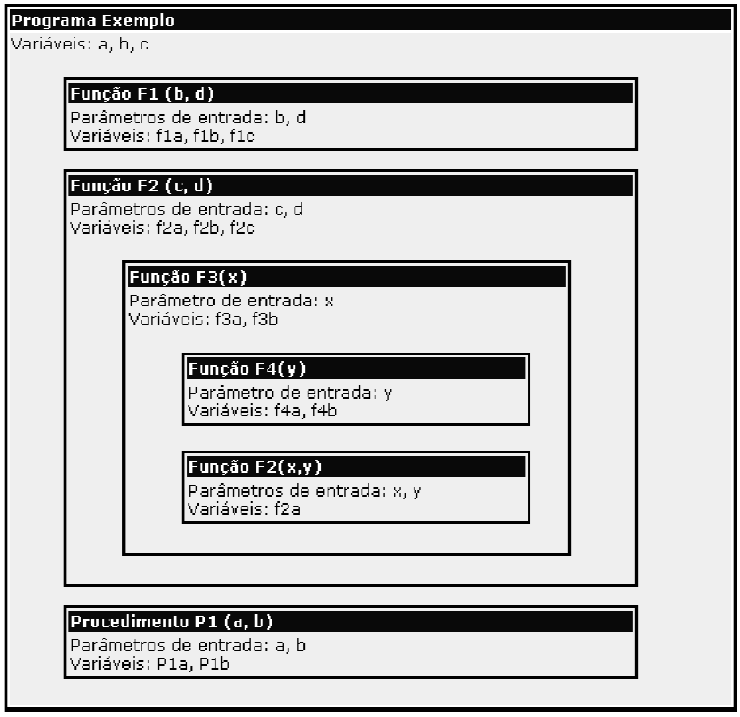
- **Critérios de qualidade de um programa (cont.):**
  - **Eficiência: refere-se à velocidade de processamento e a correta utilização da memória. Um programa deve ter performance SUFICIENTE para atender às necessidade do problema e do usuário, bem como deve utilizar os recursos de memória de forma moderada, dentro das limitações do problema.**
  - **Modularidade: consiste no particionamento do programa em módulos menores bem identificáveis e com funções específicas, de forma que o conjunto desses módulos e a interação entre eles permite a resolução do problema de forma mais simples e clara.**
  - **Generalidade: é interessante que um programa seja tão genérico quanto possível de forma a permitir a reutilização de seus componentes em outros projetos.**

Uso de funções

- **Apresenta as seguintes vantagens:**
  - **Permite evitar repetição de código, que por sua vez possibilita:**
    - **Aumentar a facilidade de manutenção do programa. Se em um código sem funções encontrarmos um erro, pode ser que esse mesmo erro ocorra em vários pontos do programa. Por outro lado, se os códigos que eram repetidos estiverem "concentrados" em uma única função, e um erro for achado nesta função, então teremos apenas um local para consertar;**
    - **Aumentar o aproveitamento de código entre programas, pois uma função bem feita (sem variáveis globais) pode ser facilmente reutilizada em outros programas.**
  - **Aumentar a legibilidade e o entendimento do programa, escondendo do programa principal detalhes que em um primeiro momento não interessam, aproximando o programa principal de seu algoritmo feito em pseudo-código (a "receita de bolo").**

Escopos de variáveis e módulos em estruturas aninhadas

- **O escopo de variáveis e módulos em estruturas aninhadas pode ser melhor compreendido pelo esquema a seguir, no qual representamos por retângulos cada módulo de um programa fictício, e pelas regras que o seguem. Neste esquema, se um módulo está dentro de outro, significa que um módulo pertence a outro, ou seja, está declarado do outro.**



Escopos de variáveis e módulos em estruturas aninhadas

- **Regras de escopos de variáveis e módulos em C:**

1. **Um módulo reconhece as variáveis e módulos que pertencem a ele e que não pertencem aos seus módulos-filhos.**
2. **Um módulo reconhece seu módulo-pai e as variáveis e módulos que ele reconhece.**
3. **No caso de haver dois ou mais módulos, variáveis ou parâmetros de entrada de mesmo nome e que seriam reconhecidos, o módulo reconhece apenas aquele que for mais "interno" na representação adotada anteriomente.**
4. **Um módulo reconhece seus próprios parâmetros de entrada.**

Escopos de variáveis e módulos em estruturas aninhadas

- **Analisando alguns casos do esquema anterior, percebemos que:**
  - **O programa Exemplo reconhece apenas as suas variáveis a, b e c, as funções F1, F2 e o procedimento P1 (tudo isso pelo item 1).**
  - **A função F1 reconhece as suas variáveis f1a, f1b e f1c (pelo item 1), seus parâmetros de entrada b e d (pelo item 4) e as variáveis a e c do programa principal (pelo item 2). Deveria reconhecer também a variável b do programa principal, mas isso causaria ambigüidade com o parâmetro de entrada b; logo, pelo item 3, apenas o parâmetro de entrada é reconhecido. A função F1 reconhece também a primeira função F2 e o procedimento P1 do programa principal (pelo item 2).**
  - **A primeira função F2 reconhece suas variáveis f2a, f2b e f2c (pelo item 1), seus parâmetros de entrada c e d (pelo item 4) e sua função F3 (pelo item 1). Também reconhece a função F1 e o procedimento P1 (pelo item 2), bem como as variáveis a e b do programa principal (também pelo item 2). O item 3 resolve a ambigüidade com relação ao c, reconhecendo apenas o parâmetro de entrada c e não a variável c do programa principal.**

Escopos de variáveis e módulos em estruturas aninhadas

- **Analisando alguns casos do esquema anterior, percebemos que (cont.):**
  - **A função F3 reconhece seu parâmetro de entrada x, suas variáveis f3a e f3b e as funções F4 e F2(segunda) (pelo item 1). Pelo item 2, ela também reconhece tudo o que a primeira função F2 reconhece, ou seja, f2a, f2b, f2c, c, d, F3 (todas pertencentes a F2) e ainda F1, P1, a e b (do programa principal). No entanto, ela não pode reconhecer F2 porque ela possui uma função F2 interna a ela, e pelo item 3 é esta função interna que é reconhecida.**
  - **A função F4 reconhece seu parâmetro de entrada y e suas variáveis f4a e f4b (pelo item 1). Pelo item 2, F4 também reconhece F3 e tudo o que ela reconhece, ou seja, x, f3a, f3b, F4, a segunda função F2 (todas internas a F3); f2a, f2b, f2c, c, d (todas internas à primeira função F2); a, b, F1 e P1 (do programa principal).**

Escopos de variáveis e módulos em estruturas aninhadas

- **Podemos considerar as variáveis e submódulos de um módulo M como sendo locais ao módulo M. Do mesmo modo, podemos chamar de globais em relação a M as variáveis e os módulos externos a M (na representação).**

- **Dado isso, a situação ideal é que cada módulo M não faça referência a variáveis que são globais em relação a ele, pois isso é o que causa a maior parte da confusão. Isso é equivalente ao que foi recomendado na aula anterior sobre não usar variáveis globais em funções.**

Passagem de parâmetros

- ***Arrays*: passar por valor ou por referência?**
  - **Quando um parâmetro é passado por valor para dentro de um procedimento (ou função), o conteúdo desse parâmetro é copiado para dentro da área de memória reservada a essa função. Assim, passamos a ter duas cópias do mesmo dado na memória (a outra cópia está na área de memória do programa principal).**
  - **Já quando passamos um valor por referência, na área de memória do procedimento fica armazenada apenas uma referência (como se fosse uma seta ou um "apontador") para a variável original, e assim não temos dados duplicados na memória.**

Passagem de parâmetros

- ***Arrays*: passar por valor ou por referência?**
  - **Essa informação pode ser muito útil no uso de *arrays*. Como *arrays* tendem a ser grandes quantidades de dados, não é desejado que esses dados estejam duas vezes na memória, o que seria um enorme desperdício de espaço. Assim, quando estivermos passando *arrays* como parâmetros, devemos pensar nos casos básicos a seguir:**
    - **Se não vamos precisar alterar o conteúdo do *array* dentro do módulo, podemos passá-lo por referência, para economizarmos espaço de memória. Passando por valor também funciona, mas teremos dados duplicados na memória.**
    - **Se vamos precisar alterar o conteúdo do *array* dentro do módulo, e essas alterações precisam ser feitas no *array* original, então também passamos o *array* por referência.**
    - **Se vamos precisar alterar o conteúdo do *array* dentro do módulo, e essas alterações não devem ser feitas no *array* original, então devemos passar o *array* por valor.**

Variáveis globais

- **Devemos evitar o uso de variáveis globais dentro de funções sempre que possível. Os principais motivos para isso são:**
  - **Reaproveitamento de funções em outros programas. Quanto mais variáveis globais uma função acessar, mais difícil será de se aproveitar essa mesma função em um outro programa. Essa dificuldade ocorre porque teríamos que garantir que as variáveis globais que essa função utilizaria estariam presentes em todos os programas em que ela fosse colocada, fato que poderia não ocorrer.**
  - **Facilidade de modificação do programa. Quando você usa variáveis globais, qualquer alteração que você faz no programa principal pode afetar todo o seu programa. Se, por exemplo, você mudar o nome de uma variável no programa principal, e ela for usada em uma função como variável global, então você precisará mudar o nome da variável também dentro da função, senão a função irá falhar.**

Variáveis globais

- **Devemos evitar o uso de variáveis globais dentro de funções sempre que possível. Os principais motivos para isso são (cont.):**
  - **Facilidade de entendimento do programa. Quando uma função acessa uma variável global, algumas vezes é difícil descobrir a quem essa variável pertence. Em programas simples, em que as funções são chamadas diretamente pelo programa principal, é difícil compreender este problema, já que as variáveis globais que uma função pode acessar são necessariamente as variáveis do programa principal.**

    No entanto, se estamos em programas nos quais uma função A chama outra função B, que chama outra função C, que chama outra função D, e a função D usa variáveis globais, então ela pode estar invocando variáveis de qualquer uma das funções C, B, A ou do programa principal. E isso pode conduzir facilmente a erros em que pensamos que estamos falando de uma variável e no fundo estamos falando de outra totalmente diferente (por exemplo, queríamos usar a variável x do programa principal, mas acabamos por usar a variável x da função C).

Critérios de avaliação em IC2

- **Dentre outros, os principais itens a serem avaliados nos programas (trabalhos e provas) são:**
  - **algoritmos (quando pedidos);**
  - **estruturação;**
  - **documentação (cabeçalho e interna ao programa);**
  - **subprogramas e parâmetros;**
  - **critérios de parada, laços e recursão;**
  - **modularidade.**