Dicas para escrever um bom código

1 Indentação

1.1 Estilos

Em C existem dois estilos de indentação mais comuns, escolha aquele que te agrada mais e nunca mude, principalmente no mesmo programa. Como pode ser visto abaixo, todas as funções e estruturas deverão respeitar o mesmo padrão.

```c
#include <stdio.h>

int main(){
    int valor;

    scanf("%d", &valor);

    if(valor < 0){
        printf("O valor eh negativo\n");
    }else{
        printf("O valor eh positivo\n");
    }

    return 0;
}
```

Listing 1: Estilo 1

```c
#include <stdio.h>

int main()
{
    int valor;

    scanf("%d", &valor);

    if(valor < 0)
    {
        printf("O valor eh negativo\n");
    }
    else
    {
        printf("O valor eh positivo\n");
    }

    return 0;
}
```

Listing 2: Estilo 2

1.2 Alguns erros comuns

```
#include <stdio.h>

int main(){
int valor;
    scanf("%d", &valor);

        if(valor < 0){
        printf("O valor eh negativo\n");
        }else{
        printf("O valor eh positivo\n");
        }
return 0;
}
```

Listing 3: Indentação inconsistente

```
#include <stdio.h>

int main(){

int valor;
scanf("%d", &valor);
if(valor < 0){
printf("O valor eh negativo\n");
}else{
printf("O valor eh positivo\n");
}
return 0;
}
```

Listing 4: Indentação ausente

Como se pode notar, tanto a inconsistência quando a ausência de indentação tornam o processo de leitura e compreensão do código complicado, mesmo em programas com algumas dezenas de linhas.

2 Blocos

Quando se trata de programas mais extensos, só a indentação não garante sua facilidade de compreensão por alguém que não o escreveu. Por isso, uma boa prática de programação é dividir seu código em blocos. O critério de separação deve ser baseado na função que aquele bloco desempenha no programa. Considere esses exemplos:

```
#include <stdio.h>
#include <stdbool.h>
#include <math.h>
int main(){
  int cat_op, cat_adj, hip;
  bool criterio;
  scanf("%d %d %d", &cat_op, &cat_adj, &hip);
  criterio = pow(cat_op,2)+pow(cat_adj,2)==pow(hip,2);
  if(criterio==true)
    printf("Sim\n");
  else
    printf("Nao\n");
  return 0;
}
```

Listing 5: Código escrito em um único bloco

```c
// Bloco de inclusao de bibliotecas
#include <stdio.h>
#include <stdbool.h>
#include <math.h>

int main(){
  // Bloco de declaracao de variaveis
  int cat_op, cat_adj, hip;
  bool criterio;

  // Bloco de leitura dos inputs
  scanf("%d %d %d", &cat_op, &cat_adj, &hip);

  // Bloco das operacoes
  criterio = pow(cat_op,2) + pow(cat_adj,2) == pow(hip,2);

  // Bloco de impressao dos outputs
  if(criterio==true)
    printf("Sim\n");
  else
    printf("Nao\n");

  return 0;
}
```

Listing 6: Código separado em blocos

Como podemos ver, no segundo exemplo temos não só uma divisão espacial, mas também uma divisão lógica que facilita tanto o entendimento quanto a manutenção do código, se for notado algum *bug* na saída do programa. Além disso, na prática, a chance de se cometer erros usando o estilo do primeiro exemplo é muito grande, pois nunca temos certeza de que parte do programa estamos trabalhando de fato, já que só existe uma.

3 Nomes de variáveis

Na hora de escrever o nome de uma variável é aconselhável usar letras minúsculas e, se houver necessidade de espaços, substitua-os por um "_". Além disso, se optar por usar um nome como x ou a (nunca use l ou o, eles lembram os número 1 e 0), não se esqueça de comentar o que aquela variável representa. Ambos os exemplos abaixo são corretos, assim cabe ao programador escolher aquele que o faz se sentir mais confortável.

```c
#include <stdio.h>

int main(){
  float comprimento;
  float largura;
  float area_retangulo;

  scanf("%f", &comprimento);
  scanf("%f", &largura);

  area_retangulo = comprimento * largura;

  printf("%.2f\n", area_retangulo);

  return 0;
}
```

Listing 7: Usando nomes explícitos

```c
#include <stdio.h>

int main(){
  float cr; // Comprimento do retangulo
  float lr; // Largura do retangulo
  float ar; // Area do retangulo

  scanf("%f", &cr);
  scanf("%f", &lr);

  // Area eh dada pelo produto entre
  // o comprimento e a largura
  ar = cr * lr;

  printf("%.2f\n", ar);

  return 0;
}
```

Listing 8: Explicitando via comentários

4 Comentários

Apenas comente as partes do seu código em que à primeira vista não são óbvias, ou seja, os comentários devem servir apenas como um complemento àquilo que o próprio código já diz, mas de uma forma simplificada. Isto fica claro no seguinte exemplo:

```c
#include <stdio.h>

int main(){
  int dia_da_semana;

  scanf("%d", &dia_da_semana);

  if(dia_da_semana < 1 || dia_da_semana > 7){
    // Se uma das condicoes eh verdade o dia nao eh valido
    printf("erro\n");
  }else{
    if(dia_da_semana == 1 || dia_da_semana == 7)
      // Se o dia for sabado ou domingo
      printf("nao util\n");
    else
      printf("util\n");
  }

  return 0;
}
```

Listing 9: Aconselhável

Por outro lado, o uso excessivo de comentários desnecessários polui o código. Abaixo temos o mesmo programa do exemplo anterior, porém descrito detalhadamente.

```
// Inclue biblioteca com as funcoes de leitura e impressao da
// linguagem c
#include <stdio.h>

// Funcao principal
int main(){
  // Declara variavel do tipo inteiro que representara um dia da
  // semana
  int dia_da_semana;

  // Faz a leitura do valor da variavel que representa um dia da
  // semana direto do terminal
  scanf("%d", &dia_da_semana);

  // Se o dia da semana for menor que 1 ou o dia da semana for
  // maior que 7
  if(dia_da_semana < 1 || dia_da_semana > 7){
    // Imprime uma mensagem de erro na tela
    printf("erro\n");

  }else{ // Caso contrario
    // Se o dia da semana for igual a 1 ou o dia da semana for
    // igual a 7
    if(dia_da_semana == 1 || dia_da_semana == 7)
      // Imprime que o dia da semana nao eh util
      printf("nao util\n");
    else // Caso contrario
      // Imprime que o dia eh util
      printf("util\n");
  }

  // Retorna sucesso!
  return 0;

}
// Fim do programa
```

Listing 10: Não aconselhável