Sugestões práticas de programação

1 Indentação

Não misture tabulações com espaços, porque tabulações tem tamanhos diferentes em programas diferentes. Se você mistura, código que parece indentado num programa parece sem indentação em outro. Para evitar a mistura, peça ao editor que mostre quais brancos são tabulações e quais são espaços.

Atenção para indicar onde os blocos terminam e acabam. Faça a indentação dos *else if* de forma a destacar os caminhos possíveis.

2 Operações aritméticas

Se houver espaço, sempre deixe espaços para destacar as operações aritméticas sendo realizadas, quanto mais externa for a operação, mais destaque ela deve ter.

Ruim	Bom
`alturaideal=mi*1000+ci*10;`	`alturaideal = mi*1000 + ci*10;`

Não mande o computador fazer a mesma conta mais de uma vez. Guarde o valor numa variável para usar de novo.

Não abuse dos parênteses, um código parecendo um cipozal atrapalha a leitura. Não queremos dizer que não pode haver parênteses desnecessários, afinal ninguém quer decorar todas as regras de precedência da linguagem, porém alguns casos são elementares (como saber que a multiplicação é feita antes da adição) e devem ser evitados.

Ruim	Bom
`altura = ((mi*(1000)) + (ci*(10)));`	`altura = mi * 1000 + ci  *10;`

Não mande o computador fazer operações aritméticas identidade. Não mande multiplicar por 1 ou somar zero. Evite mandar o computador calcular algo cujo resultado você já conhece (como multiplicar uma variável por zero).

Gaste um pouco de tempo tentando simplificar as operações aritméticas, pois com o tempo elas ficarão grandes. Se você não se acostumar que é preciso algum tempo para planejar isso, não vai conseguir fazer programas mais complicados no futuro.

Ruim:	`v = ((mi*1000)+(ci*10)+mmi)+((((mi*1000)+(ci*10)+mmi)/100)*1);`
Bom:	`v = (mi*1000  + ci*10 + mmi) * 1.01f;`

Ruim	Bom
`float a = ...;` `float b = a*101/100;`	`float a = ...;` `float b = a * 1.01f;`

3 Operações relacionais

Evite verificar se um booleano é igual a *true* pois o valor já é uma comparação em si. A verificação de igualdade com o valor *false* pode ser usada para evitar ter que inverter o valor.

Ruim	Bom
`if (Par(n) == true) ...`	`if (Par(n)) ...`

4 Declarações

Não inicialize uma variável com um valor que você sabe que não vai usar. Se o valor da variável vai ser lido, faça a leitura imediatamente após a declaração.

Ruim	Bom

`int a = 0, b = 0, c = 0;`
`cin >> a >> b;`
	`int a, b;`
	`cin >> a >> b;`
	`int c = 0;`

Ruim	Bom
`int var;` `var = a * 2 + f(x);`	`int var = a * 2 + f(x);`

Ruim	Bom

`float mi, cmi, mmi, ma, cma, mma, av, ideal, carro;`
 	
`cin >> mi >> cmi >> mmi;`
`cin >> ma >> cma >> mma;`
`ideal = conversao(mi, cmi, mmi);`
`carro = conversao (ma, cma, mma);`
	`float mi, cmi, mmi;`
	`cin >> mi >> cmi >> mmi;`
	`float ma, cma, mma;`
	`cin >> ma >> cma >> mma;`
	`float ideal = conversao(mi, cmi, mmi);`
	`float carro = conversao (ma, cma, mma);`

5 Conversão de tipos

Evite usar a sintaxe da linguagem C. A linguagem C++ trouxe novas sintaxes para conversão de tipos justamente porque as conversões de tipo eram causa frequente de erros de programação na linguagem C.

Se puder já escrever os valores das constantes com o tipo certo, melhor ainda. Exemplo: 1 é uma constante do tipo inteiro, 1.0 é uma constante do tipo *double* e 1.0f é uma constante do tipo *float*. O sufixo f informa ao compilador que a constante tem o tipo *float*. Outros sufixos existem.

Ruim	Bom
`int b = 3;` `float a = (float) b * 10;`	`int b = 3;` `float a = float(b) * 10.0f;`
	`int b = 3;` `float a = static_cast<float>(b) * 10.0f;`

6 Modularização

Quando um enunciado se refere a *receber* e *retornar*, está falando de passagem de parâmetros. Exemplo de função que recebe um inteiro e retorna um texto:

```
string funcao(int numero);
```

Ler e escrever são coisas bem diferentes de receber e retornar, são operações de interface. Exemplo de função que lê um inteiro e escreve um texto:

```
void funcao() {
    int numero;
    cin >> numero;
    ...
    cout << "texto" << endl;
}
```

Subprogramas não devem misturar interface e processamento, ou seja, ou o subprograma serve para ler informações, ou serve para escrever informações ou serve para processar informações. Essas coisas não devem ser misturadas **a não ser que o enunciado exija**.

Ruim	Bom

`void Par(int numero) {`
`  if (numero % 2 == 0)`
`  cout << "par" << endl;`
`  else`
`  cout << "impar" << endl;`
`}`
	`bool Par(int numero) {`
	`  return numero % 2 == 0;`
	`}`
	
	`int main() {`
	`  int n;`
	`  cin >> n;`
	`  if (Par(n))`
	`  cout << "par" << endl;`
	`  else`
	`  cout << "impar" << endl;`
	`}`