Programação

Como Ser Competitivo

Dica 1: Digite Rápido e Corretamente

- Não é brincadeira, é importante!
- Teste a si próprio
  - http://www.typingtest.com
  - ZEBRA – Africa's stripped horse
- Familiarize-se com a posição das teclas
  - (,), {, }, [,], <, >, ', ", &, | , !, etc;
  - Domine o *autocomplete* de algum editor.

Como Ser Competitivo

Dica 2: Identifique Rapidamente o Tipo do Problema

- *Ad Hoc* (algoritmos não tradicionais);
- Busca Completa;
- Dividir e Conquistar;
- Guloso;
- Programação Dinâmica;
- Grafos;
- Matemático;
- *String*;
- Geometria Computacional;
- Outros mais difíceis.

Como Ser Competitivo

Dica 3: Analise o Algoritmo

- Veremos o básico necessário
  - Localizar as restrições no enunciado do problema;
  - Pensar no algoritmo mais simples que funcione;
  - Realizar análises básicas que convençam que o algoritmo funciona
    - Antes de começar a codificar!

Como Ser Competitivo

Dica 4: Domine uma Linguagem de Programação

- Devemos dominar pelo menos uma linguagem de programação
  - Menos tempo olhando em referências;
  - Usar atalhos, macros, etc;
  - Usar bibliotecas sempre que possível.
- A idéia é, uma vez com a solução em mente, traduzí-la em um código livre de erros
  - E rápido.

Como Ser Competitivo

Dica 5: Dominar a Arte de Testar

- Obviamente, queremos um *Código Aceito!*
  - Nossos códigos têm que passar pelos "testes secretos" dos juízes.
- Entretanto, nem sempre é possível
  - Onde foi que eu errei?

Como Ser Competitivo

Dica 6: Prática e Mais Prática

Os Problemas

Os Problemas

- Os problemas são enunciados de forma bem humorada, em contextos fictícios, porém, de aplicação prática;
- Envolvem, dentre outros:
  - Aritmética e Álgebra;
  - Geometria computacional;
  - Manipulação de *strings*;
  - Grafos;
  - Problemas Combinatórios.

30

URI Online Judge

URI Online Judge

- O URI *Online Judge* é um site tipo o SPOJ, com enunciados de problemas e com um juiz automático
  - Brasileiro;
    - Universidade Regional Integrada do Alto Uruguai e das Missões.
  - Possui um ranking, assim como o SPOJ
    - Por programadores e por instituição.
  - Possui estatísticas sobre quantas pessoas resolveram cada problema.

URI Online Judge

- **Vantagens**:
  - Problemas categorizados por abordagem
    - *Ad hoc*, *strings*, estruturas de dados, geometria, grafos etc;
  - Problemas categorizados por nível de dificuldade
    - Nível 1 ao 9.
  - É possível saber "o quanto" sua resposta está errada
    - *Wrong Answer* (10%), *Wrong Answer* (100%), etc;
    - Depende do quanto você acertou dos casos de teste do juiz.
  - É possível saber a possível causa de erros de execução durante o julgamento.

URI Online Judge

- **Vantagens:**
  - Interface de rede social
    - *Badges*.
  - Fórum melhor do que o do SPOJ;
  - Criação de casos de teste adicionais pelo próprio site;
  - É possível salvar seus códigos do site para o *dropbox* automaticamente;
  - Possibilidade de criar competições privadas pelo site;
  - Há um plano para disponibilizar no site material e tutoriais para estudo.

URI Online Judge

- Há uma seção de problemas para iniciantes (nível 1)
  - Comecem a praticar!
  - É possível que o professor acompanhe o desempenho dos alunos pelo site e ajude nas dúvidas.

http://www.urionlinejudge.com.br/judge/en/problems/index/1

URI Online Judge

- Acesso:

http://www.urionlinejudge.com.br/

http://www.urionlinejudge.com.br/forum/

https://www.facebook.com/urionlinejudge

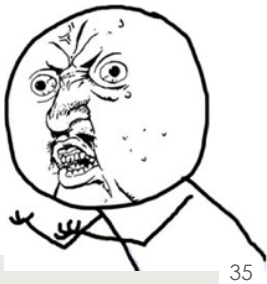
Tipos de Erros

Tipos de Erros

- O *juiz* exibe uma mensagem após a correção do código-fonte enviado;
- No caso de erro, uma mensagem específica é exibida para que o código-fonte seja corrigido e submetido novamente
  - No entanto, a mensagem nem sempre é específica sobre a localização do erro.

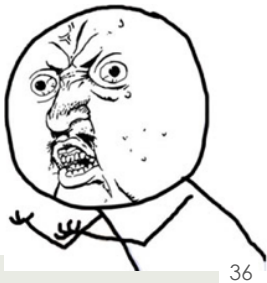
Tipos de Erros

- **Resposta errada**
  - Novamente, a bateria de testes é extensa, e embora seu programa tenha executado normalmente para os testes que você fez, há algo de errado;
  - Realize testes diferentes;
  - Verifique também os limites dados no programa, por exemplo:
  - "Leia um inteiro *n* e imprima n^2". Se *n* for 100000 e você estiver lendo *n* como inteiro e imprimindo n^2com %d vai ocorrer *overflow* na variável e provavelmente vai dar resposta errada.



Tipos de Erros

- **Resposta errada (cont.)**
  - A resposta dada pode estar em formato errado em relação ao que foi pedido;
  - Uma diferença mínima, como uma quebra de linha já é suficiente;
  - Note que, neste caso, não foi avaliado se a solução está certa ou errada.



Tipos de Erros

- **Erro em tempo de execução**
  - O programa deu pau em algum dos testes realizados;
  - Note que a bateria de testes é extensa, e embora seu programa tenha executado normalmente para os testes que você fez, há algo de errado;
  - Procure por erros de memória.

Erro em Tempo de Execução

- Não se esqueça do *return 0*!

```
int main (void) {
    //seucodigo
    return 0;
}
```

Erro em Tempo de Execução

- Não faça divisões por zero

```
int main () {
    while (n>=0) {
        /* Note que quando n==0, você fará uma divisão por zero*/
        printf("%0.2f\n",10/n);
        n--;
    }
    return 0;
}
```

Erro em Tempo de Execução

- Não acesse memória que não lhe pertence

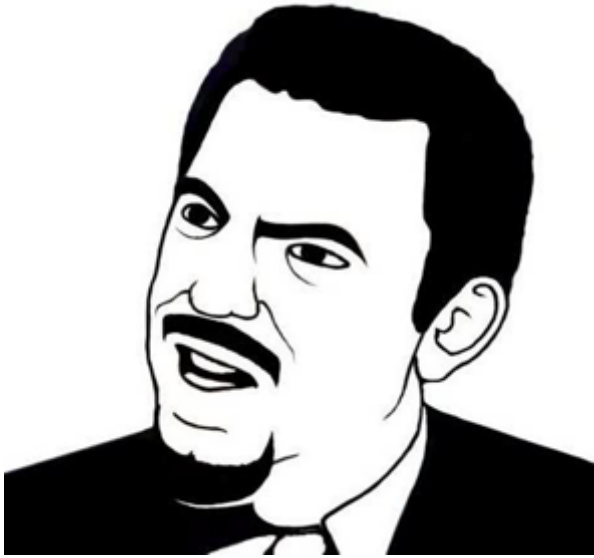
```
int main (void) {

   int v[100];

   //a posição 101 não te pertence

   printf("O elemento na posição 101 é: %d\n", v[101]);

   return 0;

}
```

Erro em Tempo de Execução

- **SIGSEGV** (signal 11) – o mais comum, "segmentation fault";
- **SIGXFSZ** (signal 25) - "output limit exceeded";
- **SIGFPE** (signal 8) - "floating point error", por exemplo, divisão por zero;
- **SIGABRT** (signal 6) – enviado pelo próprio programa; A STL (C++) faz isso de vez em quando...
- **NZEC** (non-zero exit code) – usado para linguagens interpretadas;
- ***other*** – existem outros sinais enviados que fazem com que o programa seja suspenso, todos considerados como *other*.

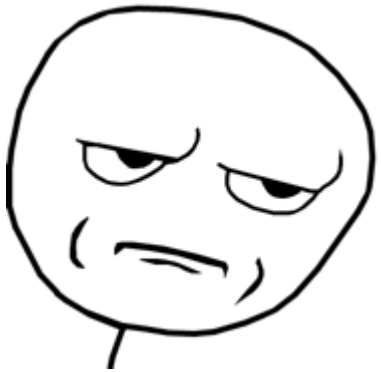
Tipos de Erros

- **Erro de compilação**
  - Simples assim.



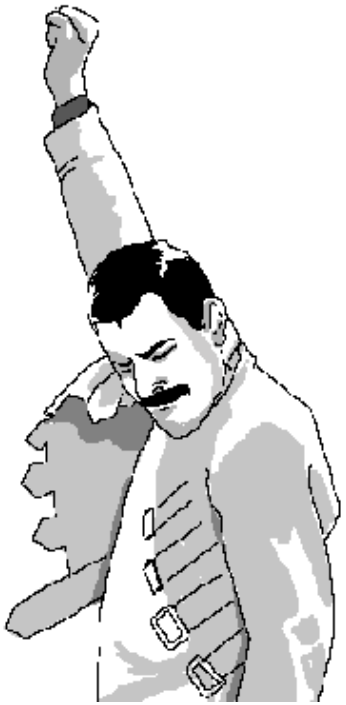
Tipos de Erros

- **Tempo limite excedido**
  - O programa demorou demais a responder;
  - Não foi avaliado se a resposta está certa ou errada;
  - Procure por procedimentos "pesados" ou *loops* infinitos.



Código Aceito

- Código aceito!



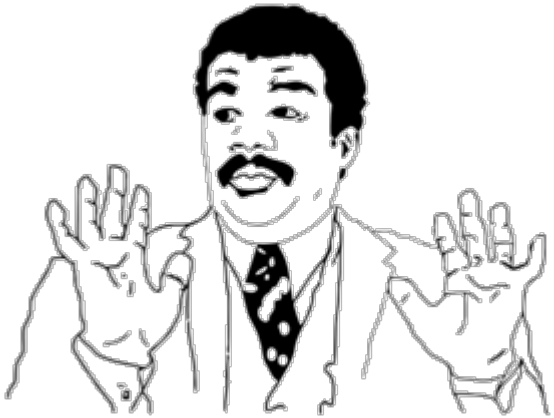
Dicas Úteis

Exemplo de Código Padrão

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cmath>
#include <cctype>
#include <algorithm>
#include <map>
#include <queue>
#include <stack>
#include <vector>
#include <iostream>
using namespace std;

const int INF = 0x3F3F3F3F;
const int NULO = -1;
const double EPS = 1e-10;

_inline(int cmp)(double x, double y = 0, double tol = EPS) {
   return (x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
}

int main(){
   int i, j;

   return 0;
}
```

Overflow e Final de Linha

- Fique atento ao intervalo dos valores da entrada e saída
  - *Overflow* aritmético.
- Não esquecer do return 0 ao final;
- Toda linha termina com '**\n**'
  - Causa *Resposta Errada*.



Entrada e Saída

- Em todos os problemas que veremos, os dados são lidos da entrada padrão e escritos na saída padrão
  - Nada de abrir arquivos ou chamadas de sistema.
- Nos testes, utilizaremos redirecionamento da entrada
  - ./programa < entrada.txt

Leitura de Dados

- Existe um aspecto importante sobre a leitura de dados
  - Até quando ler?
- Diz respeito ao término da entrada
  - Número fixo de valores, valor especial ou EOF.
- O enunciado do problema deve ser lido atentamente
  - A codificação da leitura dos dados não deve ser um processo lento.

Problemas Sugeridos

- Ver Seção “Nível Iniciante”.

Perguntas?