Universidade Federal de Viçosa – UFV
Campus de Rio Paranaíba

Dicas para Maratona de Programação

Aula 01

Prof. Thiago Pirola Ribeiro
thiago.pirola@ufv.br

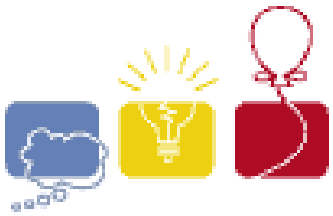
Disciplina

- **SIN343 - Desafios de Programação**
  - Ementa
    - Disciplina com intensa atividade de programação. Os problemas de programação incluem estruturas de dados, ordenação, aritmética, álgebra, combinatória, teoria dos números, backtracking, grafos, programação dinâmica, geometria computacional, dentre outros.

O que é dado por aí...

- 1. Prática com as principais linguagens de programação: tipos de dados simples e complexos, entrada e saída, manipulação de arquivos, ponteiros, cadeias de caracteres, principais bibliotecas
- 2. Algoritmos e estruturas de dados elementares: busca binária, métodos eficientes de ordenação, mediana e i-ésimo menor elemento, listas, pilhas, filas, árvores, conjuntos dinâmicos, filas de prioridade, tabelas de espalhamento
- 3. Grafos e principais algoritmos em grafos: caminho mínimo com e sem pesos, ordenação topológica, componentes fortemente conexos, fluxo máximo, emparelhamento máximo.
- 4. Exemplos de resolução de problemas com programação dinâmica: mochila booleana, soma de subconjuntos, subsequência comum máxima, subsequência crescente máxima, supersequência comum minima, alinhamento de sequências, subsequência de soma máxima
- 5. Exemplos de algoritmos gulosos: mochila fracionária, escalonamento de intervalos, árvore de Huffman, arborescência maximal de peso mínimo
- 6. Principais problemas NP-difíceis: mochila booleana, circuito hamiltoniano, clique máximo, cobertura minima
- 7. Principais algoritmos da Geometria Computacional: intersecção de segmentos, par de pontos com distância minima, localização de um ponto referente a um polígono, fecho convexo
- 8. Prática de resolução de problemas em ambientes de gerenciamento de olimpíadas de programação.

International Collegiate Programming Contest



acm International Collegiate Programming Contest

IBM event sponsor

International Collegiate Programming Contest

- Conhecido mundialmente como ICPC
- Surgiu em 1976
- Promovido anualmente pela *Association for Computer Machinery* (ACM)
- Web site: http://icpc.baylor.edu/icpc/

International Collegiate Programming Contest

- A competição possui duas etapas:
  - **Finais Regionais** – realizadas localmente, em vários países ao redor do mundo
  - **Final Mundial** – reúne os melhores colocados das competições regionais
- A final mundial de 2013 ocorrerá na cidade de São Petersburgo, Rússia.

International Collegiate Programming Contest

- A etapa regional ocorre no ano anterior à final mundial
- Por exemplo, a final mundial de 2013 irá suceder as competições regionais que ocorreram em 2012
- Apenas um time por instituição pode participar da final mundial

ETAPAS REGIONAIS

International Collegiate Programming Contest – Regionais Latino Americanas

- ACM-ICPC South America Contest within Brazil
- ACM-ICPC South America Contest outside Brazil
- ACM Mexico and Central America Programming Contest

ACM-ICPC South America Contest within Brazil

- Conhecido no Brasil como **Maratona de Programação**
- Promovido pela Sociedade Brasileira de Computação (SBC)
- Web site: http://maratona.ime.usp.br
- Diretor: Dr. Carlos Eduardo Ferreira, Professor of Computer Science (IME-USP)

Maratona de Programação

Informações Gerais

Maratona de Programação – Informações Gerais

- É um evento que nasceu das competições regionais classificatórias para as finais mundiais do concurso de programação da ACM
- É parte da regional sul-americana do ACM ICPC
- Primeira edição ocorreu em 1996
- O formato adotado para a competição é o mesmo formato do ACM ICPC
- Destina a alunos de cursos de graduação e início de pós-graduação na área de Computação

Maratona de Programação – Informações Gerais

- É dividido em duas etapas: a etapa regional, realizada em várias sedes espalhadas pelo país, e a etapa nacional, que reune os times classificados em cada uma das sedes (**two-tiered contest**)
- Um total de 30 times se classificam para a final nacional
- No máximo dois times por instituição se classificam para a final nacional
- Número de vagas para a final mundial depende do número de total de inscritos

Maratona de Programação – Informações Gerais

- Fórmula para a distribuição de 20 vagas para a final entre as sedes cadastradas

$$vagas\_na\_sede = arredonda\left(20 * \frac{número\_times\_sede}{número\_total\_times}\right)$$

- Outras 10 vagas são distribuídas entre sedes que contarem com a participação de uma instituição medalhista na competição do ano anterior

Maratona de Programação - Premiação

- A Maratona de Programação da SBC oferece medalhas aos dez primeiros colocados na fase final da competição
  - Ouro para os três primeiros
  - Prata para o quarto, quinto e sexto
  - Bronze para o sétimo a décimo lugares
- O time campeão recebe uma cópia do troféu "Maratona de Programação".

Maratona de Programação - Premiação

- O time campeão da Maratona de Programação garante participação nas finais mundiais do ACM-ICPC.
- Caso o Brasil receba outras vagas do comitê diretor da regional sul-americana da competição, estas serão ocupadas pelos primeiros colocados da final brasileira.

Maratona de Programação – Comitê Organizador

- A Maratona de Programação só acontece graças ao trabalho voluntário de muitos entusiastas da idéia
- Desde o ano 2000, o responsável pelo concurso no Brasil é Carlos E. Ferreira do Depto de Ciência da Computação do IME-USP.
- Nos primeiros anos de realização a Maratona foi organizada por Claudionor Coelho do Depto de Ciência da Computação da UFMG e por Ricardo Dahab, do IC da Unicamp

Maratona de Programação – Formato

- Times compostos por três alunos e um técnico – **three contestants and a coach**
- Cada time pode ter no máximo um reserva, e que pode substituir um competidor apenas antes do início da competição
- Cada time deve ter um nome. Fica a critério do time escolhê-lo

Nome dos Times da Primeira Fase Regional de 2012

[FACTHUS] SI FACTHUS	[UFU-FACOM] DM13
[FAZU] Bazinga!	[UFU-FACOM] Dona Flor e seus dois maridos
[FEIT] FEIT - UEMG	[UFU-FACOM] me obrigue
[FPU] FPU 1 - XGH	[UFU-FEELT] Beans team
[IFG] IFG 1	[UFU-FEELT] Biotime
[IFG] IFG 2	[UFU-FEELT] FEELT 1
[IFTM] End Of File	[UFU-FEELT] Force Staff
[IFTM] Engenharia Reversa	[UFU-FEELT] Humildemente Belos
[IFTM] IFTM 1	[UFU-FEELT] Vem Ni Mim
[IFTM] IFTM 2	[UFU-FEMEC] Paranoid Parrots
[IFTM] IFTM Uberaba	[UFV] Extreme
[IFTM] ProgAmadores	[UNIPAM] Patos de Minas
[IFTM] SIMICA	[UNITRI] Fais Freti
[ILES/ULBRA] ILES/ULBRA 1	[UNITRI] Novatos
[PITAGORAS] Begin end Team	[UNITRI] Powerpuff Boys
[PITAGORAS] Debug is on the table	[UNIUBE] UNIUBE - SI
[PITAGORAS] Not equals	[USP-ICMC] Gangnam Style
[UFMG] In rand() we trust!	[USP-ICMC] I Can Haz Balloonz
[UFMG] Naamloos	[USP-ICMC] Runtime Error
[UFMG] Starters Edition	[USP-ICMC] Senora Margarida
[UFMG] undefine_name	[USP-ICMC] Time Limit Exceeded
[UFU-FACOM] Acucar Sintatico	[USP-IFSC] Schrodingers Katze
[UFU-FACOM] BSOD	[USP-IFSC] SPAM
[UFU-FACOM] Chinelada na Orelha	

Maratona de Programação – Formato

- Apenas um computador é alocado para o time inteiro
- Período de competição de 5 horas
- Todos os times iniciam a competição com penalidade de tempo igual a zero
- O time que resolver mais problemas, com menor tempo total como critério de desempate, é o vencedor

Maratona de Programação – Formato

- Apenas é permitido acesso a materiais impressos durante o **contest**
- Para cada problema em um **problem set** o time deve implementar uma solução para resolvê-lo e submeter o código fonte aos juízes para avaliação
- As implementações devem ser em C, C++ ou Java (a critério do time)
- Cada solução aceita pelos juízes é premiada com um balão

Maratona de Programação - Formato

- A cada solução aceita, o tempo decorrido desde o início da competição até a submissão da solução correta é somado ao tempo total do time
- A cada solução rejeitada, uma penalidade de 20 minutos é somada ao tempo total do problema
- Quando um problema for solucionado pelo time, todas as penalidades acumuladas associadas ao problema também serão somadas ao tempo total do time

Maratona de Programação – Formato

- Cada problema contém:
  - Informações para contextualização (**background**)
  - O enunciado do problema
  - Informações sobre a entrada (**Input**)
  - Informações sobre a saída (**Output**)
  - Exemplo de entrada (**Sample Input**)
  - Exemplo de saída (**Sample Output**)

Maratona de Programação – Formato

- Informações para contextualização (**background**)
  - Contém informações gerais que servem apenas para dar sentido ao problema, encaixando-o no que poderia ser uma situação real no qual o mesmo se aplica.

Maratona de Programação – Formato

- Enunciado do problema
  - Contém as intruções sobre que tipo de entrada será fornecida à solução que op time propor e que tipo de resposta é esperada para a entrada dada.
  - Ex: Para este problema, seu trabalho é fazer um programa que, dados dois número N e M, retorne a soma de todos os número inteiros entre N e M, inclusive.
- Geralmente o enunciado do problema e **background** não são divididos no problema, o que não implica em dificuldade adicional

Maratona de Programação – Formato

- Informações sobre a entrada (**Input**):
  - Informações sobre a formatação da entrada
    - Ex: Cada caso de teste é composto por dois números, N e M, sendo, respectivamente….
  - Contém informações sobre as variáveis que devem ser lidas pela solução submetida
    - Ex: N é o número inicial, M é o número final
  - Quais as limitações das informações de entrada
    - Ex: 1 < N < 1000000, 2 < M < 2000000
  - Informações sobre como terminar a leitura do caso de teste
    - O fim de um caso de teste é indicado por N = M = 0.

Maratona de Programação – Formato

- Informações sobre a saída (**Output**)
  - Contém informações sobre como a saída deverá ser formatada
    - Ex: Para cada caso de teste, seu programa deverá implimir uma linha “Instance #i:”, onde *i* é o caso de teste corrente, seguida de uma linha com um número, indicando o somatório dos números da entrada. Cada número deve ser impresso em uma única linha.

Maratona de Programação – Formato

- Exemplo de entrada (**Sample Input**)
  - Contém um pequeno exemplo sobre o que é uma entrada válida para o problema
  - Ex:

```
1 8

4 100

0 0
```

Maratona de Programação - Formato

- Exemplo de saída (**Sample Output**)
  - Contém a saída para o exemplo de entrada (**Sample Input)** dado
  - Ex:

Instance #1:

36

Instance #2:

5044

Maratona de Programação - Formato

- Os juízes possuem **datasets** que são utilizados para testar a solução submetida
- Esses **datasets** contém instâncias de casos de testes bastante diferentes dos exemplos contidos nos problemas do **problem set**

Maratona de Programação – Formato

- Os juízes podem dar uma das seguintes respostas a uma solução submetida por um time:
  - Yes
  - No – Wrong Answer
  - No – Presentation Error
  - No – Time Limit Exceeded
  - No – Runtime Error
  - No – Compile Error

Maratona de Programação – Formato

- Nenhuma outra informação será dada ao time caso algum problema tenha sido detectado na solução submetida. A única informação que o time obterá dos juízes, acerca da correção de uma solução, será uma das respostas mencionadas

Maratona de Programação - Formato

- **Yes**
  - Resposta dada pelo juíz a uma solução que passou em todos os testes de caso, dentro do tempo limite designado para que a solução submetida pelo time retornasse as respostas
  - O time ganha um balão ao receber um **Yes** para uma solução submetida

Maratona de Programação – Formato

- **No – Wrong Answer**
  - A solução submetida pelo time não retornou a resposta correta para um ou mais dos testes de casos contidos no **dataset** associado ao problema
  - O juíz não irá informar em qual teste de caso a solução submetida pelo time falhou, ou dar qualquer dica sobre a falha

Maratona de Programação – Formato

- **No – Presentation Error**
  - A saída da solução não obedeceu algumas regras estabelecidas na descrição do problema (**Output**)
  - Espaços a mais (ou a menos), bem como quebras de linhas indevidas ou faltando são os principais responsáveis por essa rejeição
  - A solução submetida pelo time está 99% correta: falta apenas verificar a formatação de saída
  - Por estar apenas 99% correta, o time não ganha o balão

Maratona de Programação – Formato

- **No – Time Limit Exceeded**
  - Significa que a solução submetida pelo time demorou mais tempo processando a entrada dos juízes do que os juízes designaram
  - Cada problema possui um tempo limite associado
  - O tempo limite é calculado de acordo com a complexidade do problema, e multiplicado por um fator que reflete as capacidades do hardware no qual o problema será corrigido, linguagem utilizada, etc.

Maratona de Programação – Formato

- **No – Runtime Error**
  - Significa que, durante o processamento das entradas, o programa abortou a execução (**abnormally**)
  - Pode ocorrer devido a uma exceção não tratada, NULL pointer, divisão por zero, devido ao uso de chamadas inválidas, etc.
  - Os juízes não informarão ao time por que a falha (**Runtime Error**) ocorreu

Maratona de Programação – Formato

- **No – Compile Error**
  - O código fonte da solução, submetida pelo time, não compilou na máquina dos juízes
  - Pode ocorrer devido ao time ter dito que o código fonte da solução submetida estava em C, quando na verdade estava em Java (**common mistakes**)
  - Pode ocorrer quando o time muda um detalhe no código fonte e esquece de testar, antes de re-enviar a solução
  - Quando o time esquece de importar os headers necessários.
    - Ex: #include <esqueci.h>

Maratona de Programação - Formato

- O time não ganhará mais pontos por problemas 90% resolvidos, por elegência na implementação do algoritmo, ou por algoritmos extremamente eficientes

Maratona de Programação - Formato

- Durante a competição, o time terá direito a:
  - Imprimir o código fonte de um problema (**tasks**)
  - Solicitar esclarecimentos sobre um problema (**clarifications**)
  - Pedir ajuda (**S.O.S.**)
- Isso tudo é solicitado através do ambiente computacional

Maratona de Programação – Ambiente Computacional

- O ambiente para submissão de problemas utilizado na final nacional é o BOCA – BOCA Online Contest Administrador
  - Desenvolvido por Cássio Polpo de Campos (PUC-SP)
  - http://maratona.ime.usp.br/manualBOCA.html
- O sistema operacional utilizado é o Maratona Linux
  - Contém todas as ferramentas necessárias para o time durante a competição
  - http://maratona.ime.usp.br/dicasLinux.html
  - Sistema operacional Linux adaptado por Cássio Polpo de Campos para competições de programação

Maratona de Programação - Regras

- No máximo três times por instituição de ensino ou departamento
- Todos os times de uma mesma instituição ou departamento devem competir na mesma sede
- No máximo dois times de uma mesma instituição ou departamento se classificam para a final nacional
- No máximo um time de uma instituição de ensino se classifica para a final mundial
- Um time só se qualifica à final se resolver no mínimo dois problemas na regional

Maratona de Programação - Regras

- Prazo para a inscrição de times deve ser obedecido
- Pedido de inscrição será atendido por ordem de chegada
  - uma sede comporta no máximo 50 times
- O coach de uma instituição pode ser o coach de todos os times
- O coach deve ser um docente, ou então ser indicado por um representante da instituição, mediante preenchimento de um formulário

Maratona de Programação - Regras

- Qualquer comunicação entre o coach e o time, durante a competição, é proibida
- Bem como a utilização de qualquer dispositivo eletrônico e material on-line
- Times compostos por três competidores oficiais mais um reserva
  - O reserva apenas pode substituir um competidor antes do início da competição
- Há certas condições que devem ser satisfeitas por todos os integrantes do time (exceto coach) para um time ser considerado elegível

Maratona de Programação - Regras

- Os membros do time devem ter participado de no máximo uma final mundial do concurso da ACM e…
- De no máximo 4 (quatro) regionais sul-americanas do concurso e…
- E (adaptado para as competições de 2006)…
  - Ter iniciado seus estudos universitários no ano de 2002 ou anos posteriores (a contar do início do primeiro curso universitário do aluno)
  - Ou ter nascido em 1983 ou anos posteriores

Maratona de Programação - Regras

- Se (adaptado para as competições de 2006)
  - o competidor já participou de duas finais mundiais ou
  - o competidor já participou de cinco regionais ou
  - (o competidor iniciou seus estudos após o fim do ensino médio no ano 2001 ou anos anteriores e o competidor nasceu em 1982 ou anos anteriores)
- Então
  - o competidor não é elegível
- se não
  - o competidor é elegível

Maratona de Programação – Como se Preparar?

- Para o competidor
  - Escolher a linguagem.
  - Noções de Complexidade de Algoritmos
  - Dominar as estruturas de dados básicas.
  - Dominar Entrada e Saída
  - Ficar à vontade com Recursão e Backtracking.
  - Aprender Grafos.
  - Aprender Programação Dinâmica.
  - Aprender técnicas para resolução de problemas envolvendo geometria computacional

Maratona de Programação – Como se Preparar?

- Para o time
  - Conhecer as habilidades e fraquezas de cada um dos colegas
  - Treinar simulando as mesmas condições que a competição real (um computador, caderno de problemas, tempo limitado, etc.)

Maratona de Programação – Como se Preparar?

- Há sites na Internet com Juízes Online (**Online robot judges**), que podem ser utilizados para treinamento individual ou do time
  - http://www.programming-challenges.com
  - http://acm.timus.ru (Ural State University)
  - http://acm.uva.es (Universidade de Valladolid)
  - http://livearchive.onlinejudge.org/
  - http://br.spoj.pl (*Sphere Online Judge* Brasil)
  - http://www.urionlinejudge.com.br

Universidade Federal de Viçosa – UFV
Campus de Rio Paranaíba

Dicas para Maratona de Programação

info@lto

Aula 02

Prof. Thiago Pirola Ribeiro
thiago.pirola@ufv.br

Dicas para a Maratona

Maratona de Programação - Dicas

- A Maratona de Programação é uma competição de criação e implementação de Algoritmos.
  - Tratamento de erros (quando o escopo da entrada é bem definido) e Interfaces com o usuário são detalhes com os quais os competidores não precisam se preocupar (e não devem)
- É melhor escrever nomes de variáveis e funções sugestivos do que escrever comentários no código

Maratona de Programação - Dicas

- O caderno de problemas possui problemas fáceis, moderados e difíceis. Se um problema parecer de difícil implementação, considere abordar outro problema
- Entretanto, alguns problemas podem parecer difíceis, e outros podem parecer fáceis
  - Ex: Dados dois polígonos, encontrar a área de intersecção. Simples?

Maratona de Programação - Dicas

- Certifique-se de que o problema realmente foi entendido, amntes de partir para a implementação
- Certifique-se de que o programa está correto, antes de submeter ao juíz. Uma penalidade pode custar mais caro que 10 minutos verificando o problema
  - Após tentar as entradas e saídas de exemplo, crie instâncias de entrada que exploram os valores limites especificados no problema

Maratona de Programação - Dicas

- Procure sempre evitar o processamento desnecessário de dados, tais como a expansão de todos os ramos de uma árvore.
  - Típico erro quando se aprender a programar: testar (i == j) para se varrer os elementos da diagonal principal de uma matriz
- Procure definir papeis para cada um dos integrantes do time durante a competição
  - Times cujos competidores brigam pelo terminal durante a competição não vão a lugar nenhum
- Há apenas um computador disponível, portanto, procure utilizar o mesmo apenas para digitação de soluçoes, e maximizar o uso do mesmo

Maratona de Programação - Dicas

- Caso a sua solução para um problema receba uma das respostas **No – Wrong Answer** ou **No – Time Limit Exceeded** por duas vezes seguidas, considere entregar o problema a outra pessoa para lê-lo, entendê-lo e implementá-lo

Maratona de Programação - Dicas

- Verifique periodicamente o **standing**. Ele conterá informações valiosas sobre problemas fáceis e difíceis. Se vários times acertaram um problema que seu time ainda não estudou, seria o caso de considerar estudar esse problema
- Entretanto, evite implementar o “algoritmo da formiga”. Muitas vezes, os times saem resolvendo problemas na ordem em que aparecem no caderno, ao invés de utilizar a baixa complexidade do problema como critério. Assim, seu time pode ter a falsa sensação de que um problema é fácil, se seguir os passos de outro time.

Maratona de Programação - Dicas

- Adicione um loop para estouro de tempo ou então implemente uma divisão por zero em trechos onde você imagina que seu programa possa estar falhando, caso seu time receba sucessivas vezes a resposta **No – Wrong Answer**. Seu time obterá um bit de informação, porém em troca de uma penalidade de 20 minutos
- Não adianta iniciar um problema nos minutos finais da competição, uma vez que geralmente os problemas que faltam ser resolvidos são de compexidade maior. É melhor modelar alguns problemas no início ou utilizar os minutos finais para a depuração de problemas já implementados, mas que ainda não foram aceitos pelo juízes

Maratona de Programação - Dicas

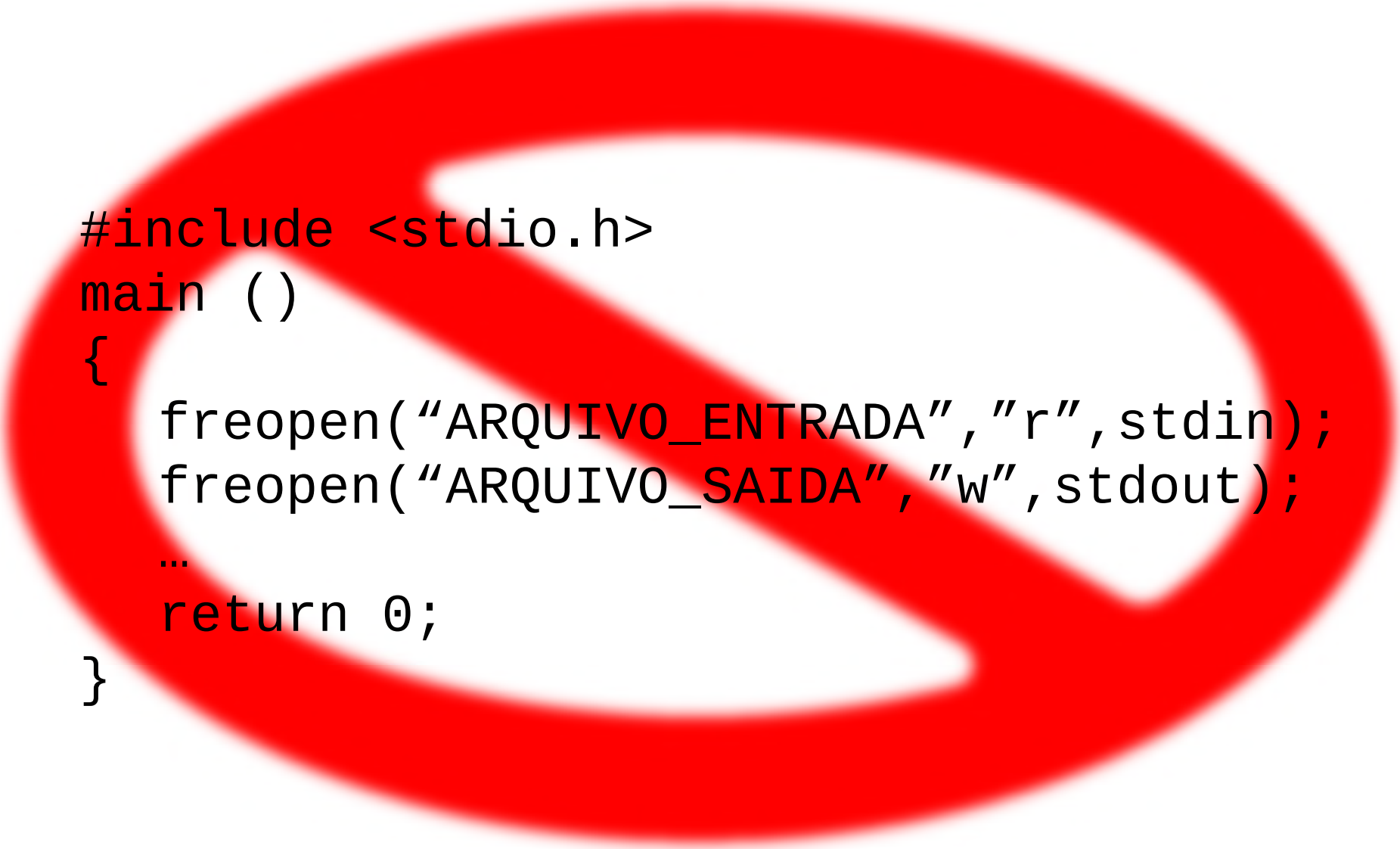
- Procure adotar uma estratégia para seu time
  - Estratégia simples: cada integrante do time pega um problema para resolver
  - Terminal Man: uma única pessoa fica encarregada do terminal, enquanto os outros dois elaboram os algoritmos e casos de testes
  - Think Tank: todos os problemas são modeloados pelo time, para finalmente serem definidos os problemas que serão implementados
- Estratégias possuem pontos positivos e negativos, e nem sempre são adequadas a todos os times. Procure elaborar uma estratégia que explore ao máximo as capacidades individuais

CODIFICANDO

E/S de dados

```
#include <stdio.h>
main ()
{
   freopen(“ARQUIVO_ENTRADA”,”r”,stdin);
   freopen(“ARQUIVO_SAIDA”,”w”,stdout);
   …
   return 0;
}
```

E/S de dados



```
#include <stdio.h>
main ()
{
   freopen(“ARQUIVO_ENTRADA”,”r”,stdin);
   freopen(“ARQUIVO_SAIDA”,”w”,stdout);
   …
   return 0;
}
```

E/S de dados

```
#include <stdio.h>
main ()
{
   freopen(“ARQUIVO_ENTRADA”,”r”,stdin);
   freopen(“ARQUIVO_SAIDA”,”w”,stdout);
   …
   return 0;
}
```

!!!NÃO UTILIZE!!!

- Bibliotecas:
  - Não utilize nada que não esteja no C ou C++ Padrão (ANSI):
    - string.h
    - conio.h
    - etc.

!!!NÃO UTILIZE!!!

- Pacotes / Packages
- Funções / Function / Procedure

!!! EVITE !!!

- Operadores ++ ou --
  - Utilize instruções separadas, pois podem ocorrer problemas no momento da execução.
  - *p++
  - Ao invés de (p+5) use p[5].

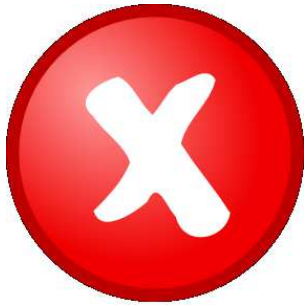
!!! ATENÇÃO !!!

```
return (x*y)+Func(t)/(1-s);
```

```
temp = func(t);
RetVal = (x*y) + temp/(1-s);
return RetVal;
```

!!! ATENÇÃO !!!

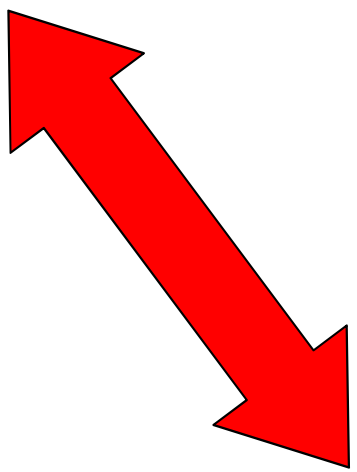
```
Temp = ((x*8-111)%7);
if (5<Temp)
   return y;
Else
   return 8-x;
```



```
return (((x*8-111)%7)>5) ? y : 8-x;
```

Condicional

```
if (a < b)
   x = a;
else
   x = b;
```



```
x = (a<b ? a : b);
```

Enunciado

- leitura **cuidadosa** (das entrelinhas . . .)
- (**entrada**) supor apenas o que está escrito
  - i não era necessariamente menor que **j**
  - terminar a leitura com fim de arquivo

```
1   while(scanf("%d %d", &i, &j) = = 2)
2   {
3   [...]
4   }
```

  - várias instâncias
- (**saída**) seguir as instrução literalmente

  (número de espaços entre . . .)

  "*output **i**, **j** in the same order in which they appeared*"

Compilação

- Evite Warnings!
- gcc -o prog prog.c -Wall -ansi -pedantic -O2

```
1 int main( ) {
2    int i, j;
3    printf("%d", j);
4    return 0;
5 }
```

```
In function ‘main’:
warning: unused variable ‘i’
warning: ‘j’ might be used uninitialized in this function
```

Testes

- fazer testes do tipo que sabemos “**resolver na mão**”
- testar com o exemplo do enunciado
- **não se iluda** com os teste do enunciado
- **não se iluda** com os teste do enunciado
- testar com “casos extremos”
- **testar = tentar “quebrar” o programa**

Exemplo

```
int main(void){
   float f;
   f = 0.8;
   printf("f = %f\n", f);
   if (f == 0.8)
      printf("f == 0.8\n")
   else
      printf("f != 0.8\n");
   return 0;
}
```

Saída

```
f == 0.800000
f != 0.8
```

Erros

"*Errar é humano, mas colocar a culpa dos próprios erros no outros é mais humano ainda.*"

Anônimo

"*Testes mostram a presença de erros, mas **não** a ausência.*"

Edsger W. Dijkstra

"***Iniciantes** tendem a colocar a culpa dos erros no compilador, na biblioteca, no mau tempo, . . .*
***Programadores experientes** gostariam de ser iniciantes para a ter quem culpar, além deles mesmos . . . "*

The Practice of Programming

Kerningham e Pike

Leitura

- **scanf**

```
int scanf(const char *formato, . . . )
```

%d **int**

%ld **long int**

%u **unsigned int**

%c **char**

%f **float**

%lf **double** (no printf use %f)

%s **char[ ]** = vetor de caracteres = *string*

scanf

```
int scanf(const char *formato, . . . )
```

%[exp] **char[ ]**

coloca no vetor enquanto o caractere lido estiver em exp

%[^exp] **char[ ]**

coloca no vetor até ser lido um caractere em exp

espaço em branco: elimina **“ ”**, **tab** e **\n**

caractere ‘normal’: lê o caractere e o joga fora

fgets

```
int fgets(char *s, int n, FILE *fluxo)
```

Lê no máximo n - 1 caracteres no vetor s, parando se um caractere *newline for encontrada; o caractere newline* ***é*** **incluído** no vetor, que é terminado por '\0'. fgets retorna s ou NULL se houver um fim de arquivo ou erro.

`stdin` = leitura da entrada padrão.

getchar

`int getchar(void)`

Leitura de um único caracterte da entrada padrão.

`getchar` é equivalente a **`getc(stdin)`**

Saída

```
int printf(const char *formato, . . . )
```

é semelhante ao

```
int scanf(const char *formato, . . . )
```

```
int putchar(char ch)
```

é semelhante ao

```
int getchar(void)
```

RELEMBRANDO...

RELEMBRANDO...

1. Apenas codifique depois de finalizar o seu algoritmo.
2. Crie dados de teste para os casos complicados.
3. Codificar a rotina de entrada e teste-a (escrever rotinas de saída extra para mostrar os dados).
4. Codificar a rotina de saída e testá-la.
5. Escreva apenas as estruturas de dados necessárias.

RELEMBRANDO...

6. Refinamento gradual: escrever comentários descrevendo a lógica do programa.
7. Escreva o código e faça depuração, a cada momento.
8. Faça o código funcionar e verifique sua corretude (use os casos de testes triviais ou dados no problema).
9. Tente quebrar o código - casos especiais de uso para acertar o código.

DURANTE A COMPETIÇÃO...

Na competição...

1. Força Bruta quando você puder: algoritmos de força bruta tendem a ser mais fáceis de implementar.
2. O simples é inteligente! (Mantenha-o simples / Mantê-lo curto e simples).
3. Dica: foco nos limites (especificados na declaração do problema).
4. Desperdício de memória: se tornar sua “vida” mais fácil, faça-o (queime memória para ter velocidade).
5. Não exclua a sua saída da depuração extra, comente-a.

Na competição...

6. Otimizar progressivamente, tanto quanto for necessário.
7. Mantenha todas as versões de trabalho.
8. Código para depurar:
    a. espaço em branco é bom,
    b. usar nomes de variáveis significativas,
    c. não reutilizar variáveis (não estamos fazendo engenharia de software aqui)
    d. refinamento gradual
    e. comentários antes do código.

Na competição...

9. Evite ponteiros, se você puder.
10. Evite memória dinâmica: alocar tudo estaticamente.
11. Tente não usar ponto flutuante, se for necessário, coloque tolerâncias em todos os lugares (nunca teste igualdade)
12. Comentários sobre os comentários:
    a. “Não escreva poesia”, apenas notas curtas.
    b. Explique em alto nível a funcionalidade:

       ++i; /* aumentar o valor de i */ é pior do que inútil.

    a. Explique “os insights” de código.
    b. Delimitar e documentar as seções funcionais.

Universidade Federal de Viçosa – UFV
Campus de Rio Paranaíba

Dicas para Maratona de Programação

info@lto

Aula 03

Prof. Thiago Pirola Ribeiro
thiago.pirola@ufv.br

GDB

The GNU Project Debugger

GDB

- A primeira versão foi publicada em 27/07/1999
- Atualmente o GDB está na versão 7.5 (17/08/12)
- Suporta as linguagens: C, C++, D, Go, Objective-C, Fortran, Java, OpenCL C, Pascal, assembly, Modula-2, and Ada.

GDB

- Compile o programa com a opção -g
- **gcc -g -o prog prog.c**

  -g : gera informações para o debugger

  -o : nome do arquivo de saída (opcional)

  prog.c : nome do arquivo com o código fonte

Outros parâmetros do gcc:

-Wall : Mostra todos os warnings.

-**pedantic** : ativa alguns avisos que existem de acordo com o padrão C, mas que são desativados normalmente por extensões do gcc.

-**ansi** : compila seu programa estritamente de acordo com o padrão ANSI, desativando qualquer extensão específica do gcc.

-**w**: Elimina mensagens de warning.

-**I/path** : Acrescenta path include.

-**l/path/lib** : Inclue biblioteca (lib).

-**O** : Optimiza o código (-O1,-O2,-O3).

GDB

- Para executar o seu programa faça:

prompt:~$ gdb prog

```
GNU gdb (Ubuntu/Linaro 7.4-2012.02-0ubuntu2) 7.4-2012.02
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/thiago/prog...done.
(gdb)
```

```
prompt:~$ gdb prog
GNU gdb (Ubuntu/Linaro 7.4-2012.02-0ubuntu2) 7.4-2012.02
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/thiago/prog...done.
(gdb)
```

COMANDOS

Comandos

- **help:** Lista as classes de comandos

```
(gdb) h
List of classes of commands:
[...]
```

- **run:** Executa o programa que pode receber um arquivo de dados como entrada:

```
(gdb) r < dados.txt
```

- **b ou break:** Coloca um *breakpoint um uma linha ou função:*

```
(gdb) break main
Breakpoint 1 at 0x804843d: file prog.c, line 5.
(gdb) b 4
Note: breakpoint 1 also set at pc 0x804843d.
Breakpoint 2 at 0x804843d: file prog.c, line 4.
```

Comandos

- **n ou next :** Executa o próximo passo do programa:

```
(gdb) n
10 while (n != 0)
(gdb) n
23 total = 0;
```

- **until:** Executa até a linha especificada:

```
(gdb) u 12
```

- **continue:** Executa até o próximo *breakpoit ou até o fim* do programa:

```
(gdb) c
Continuing.
```

Comandos

- **step:** Semelhante ao *next só que "entra" em funções:*

```
(gdb) s
24 for (i = 0; i < n; i++)
```

- **print:** Mostra o conteúdo de uma variável:

```
(gdb) p i
$6 = 0
```

- **display:** Mostra o conteúdo de uma variável a cada passo:

```
(gdb) display i
```

Comandos

- **set variable**: Define um valor para uma variável:

```
(gdb) set variable i=20
(gdb) display i
2: i = 20
(gdb)
```

- **watch:** Para a execução do programa e mostra o conteúdo de uma variável sempre que ela é alterada:

```
(gdb) watch i
Hardware watchpoint 2: i
```

- **quit:** Encerra a execução do gdb:

```
(gdb) quit
The program is running. Exit anyway?(y or n)
```

Links úteis

- Site do projeto
  - http://www.gnu.org/software/gdb/
- Documentação
  - http://sourceware.org/gdb/current/onlinedocs/