Programação

IFMG CODAAUT

Prof. Marco Antonio M. Carvalho

UFOP
Universidade Federal de Ouro Preto

PROEX
Pró-reitoria de extensão da UFOP

decom
departamento de computação

INSTITUTO FEDERAL
MINAS GERAIS

Lembretes

- **Lista de discussão**
  - Endereço:
    - programaacao@googlegroups.com
  - Solicitem acesso:
    - http://groups.google.com/group/programaacao
- **Página com material dos treinamentos**
  - http://www.decom.ufop.br/marco/extensao/obi/
- **Repositório online de problemas das edições passadas da OBI**
  - http://br.spoj.com/problems/obi/sort=-7
- **Moodle**
  - http://programaacao.net.br/login/index.php

Avisos

Na aula de hoje

- Vetores
  - Ordenação.
- Representação de Tipos;
- Problemas *Ad Hoc*.

Vetores

Vetores

- Um vetor é um conjunto de variáveis
  - Do mesmo tipo;
  - Referenciadas pelo mesmo identificador (nome);
  - Cada variável é diferenciada das outras por um número, chamado de índice.

```
vector<int> nota[9];
```

Índice	0	1	2	3	4	5	6	7	8
Valor	6.5	4.3	7.8	9.8	10.0	9.8	8.7	10.0	10.0

Vetores

- Vetores são variáveis compostas homogêneas unidimensionais;
- Sintaxe

vector<tipo> identificador(dimensao);

- Em que:
  - **tipo**: é um tipo da linguagem C++;
  - **identificador**: é um identificador válido em linguagem C++;
  - **dimensao**: é o número de posições, um número inteiro, que determina quantos valores poderão ser armazenados no vetor.
- Note que é necessário incluir a biblioteca `<vector>`.

Vetores

```
#include <vector>
using namespace std;
int main()
{
    vector<int> numeros(50);
    vector<char> letras(10);
    vector<float> notas(14);
    vector<double> aproximacoes(999);

    return 0;
}
```

Vetores

- Somente um identificador é utilizado
  - A distinção entre as posições é feita pelo índice.
- O índice permite o acesso direto a uma determinada posição
  - Não é necessário percorrer todas as outras até chegar na desejada.

Vetores

```
int main()
{
    vector<int> numeros(50);

    numeros[4] = 1;
    numeros[10] = 2;
    numeros[49] = 3;

    return 0;
}
```

Vetores

Atenção!

- A primeira posição de um vetor sempre é a posição **zero**;
- A última posição é sempre **tamanho-1**;
- É possível tentar escrever/ler uma posição indevida no vetor
  - O compilador não faz esta verificação;
  - Causa erro de execução!

```cpp
int main()
{
    vector<int> numeros(5);//o vetor possui 5 posicoes

    numeros[0] = 1;//primeira posicao
    numeros[1] = 2;
    numeros[2] = 3;
    numeros[3] = 4;
    numeros[4] = 5;//última posicao
    numeros[5] = 6;//erro!
    numeros[6] = 7;//erro!
    return 0;
}
```

Vetores

- As posições de um vetor podem ser utilizadas em expressões algébricas e lógicas
  - Cuidado para o tipo resultante da expressão não ser diferente do tipo do vetor.

```
int main()
{
    vector<int> numeros(5);

    numeros[0]+=numeros[1];

    cout<<numeros[3]<<endl;

       numeros[1]=numeros[2]*numeros[3];

      if(numeros[3]>=numeros[4])
       numeros[3]++;
    return 0;
}
```

Vetores e Estruturas de Repetição

- Para percorrer ou preencher um vetor, utilizamos estruturas de repetição
  - Qualquer uma delas;
  - Porém, a instrução *for* é a mais utilizada, por causa do seu contador, que pode ser utilizado como índice.

```
#define TAMANHO 5

int main()
{
   vector<int> vetor(TAMANHO);

   printf("Digite %d números\n", TAMANHO);

   for(int i=0; i<TAMANHO; i++)
      cin>>vetor[i];

   for(int i=0; i<TAMANHO; i++)
      cout<<vetor[i]<<endl;

   return 0;
}
```

Ordenação

- Frequentemente nos deparamos com a tarefa de ordenar valores numéricos armazenados em um vetor;
- Felizmente, temos um algoritmo de ordenação eficiente já implementado na biblioteca padrão do C++;
- Por padrão, o algoritmo ordena um vetor de modo crescente, mas podemos alterá-lo para ordenar de modo decrescente.

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
int main(){
    vector<int> vetor(TAMANHO);

    printf("Digite %d números\n", TAMANHO);
    for(int i=0; i<TAMANHO; i++)
        cin>>vetor[i];

    //ordena de forma crescente
    sort (vetor.begin(), vetor.end());
    return 0;
}
```

```
//função auxiliar
bool decrescente (int i,int j){
       return (i>j);
}

int main(){
   vector<int> vetor(TAMANHO);

   printf("Digite %d números\n", TAMANHO);
   for(int i=0; i<TAMANHO; i++)
      cin>>vetor[i];

   //ordena de forma decrescente
   sort (vetor.begin(), vetor.end(), decrescente);
   return 0;
}
```

Mandamentos do Uso de Vetores

1. Não alocarás dinamicamente;
2. Declararás o vetor com tamanho **maior** do que o limite máximo do problema;
3. Limparás o vetor antes de usar;
4. Não cobiçarás a memória do próximo.

Problemas Selecionados

- http://br.spoj.com/problems/PUSAPO11/
- http://br.spoj.com/problems/FLIPERAM/
- http://br.spoj.com/problems/MINADO12/
- http://br.spoj.com/problems/ELEICOES/
- http://br.spoj.com/problems/IMPEDIDO/

Representação de Tipos

Representação de tipos

- Mantenha a simplicidade
  - Nada de matar formiga usando tanques de guerra.
- Familiarize-se com os tipos de dados primitivos
  - A partir deles quase tudo pode ser criado e resolvido.
- Será que a variável comporta os valores dos problemas?

Mandamentos do *Overflow* Aritmético

1. Não dividirás por zero;
2. Realizarás testes dos limites dos valores;
3. Compararás números reais usando cmp():

```
const double EPS = 1e-10;
//retorna 1 caso x>y
//retorna 0 caso x==y
//retorna -1 caso x<y
int cmp(double x, double y = 0, double tol = EPS) {
   return (x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
}
```

Representação de Tipos

Tipo		Faixa
Caracteres	*char*	-128 a 127
Inteiros	*int*	–2.147.483.648 a 2.147.483.647
	unsigned int	0 a 4.294.967.295
	long	–2.147.483.648 a 2.147.483.647
	unsigned long	0 a 4.294.967.295
	long long	–9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
Ponto Flutuante	*float*	3.4E +/- 38 (precisão de 6 dígitos)
	double	1.7E +/- 308 (precisão de 15 dígitos)

Representação de Tipos

- Note que o tamanho dos tipos não é definido pela linguagem
  - Na verdade, depende da arquitetura utilizada
    - 16 bits, 32 bits, 64 bits...
- Para maior precisão, existem estruturas e bibliotecas para números de alta precisão
  - Por exemplo, a *bignum*;
  - Com isso surge a *aritmética de inteiros de alta precisão*;
    - Mas isso é assunto para outra aula...

Representação de Tipos

- Estar atento aos limites de cada tipo é especialmente importante para evitar erros;
- Na descrição da entrada de cada problema, é informado o intervalo de valores que podem ser utilizados na entrada
  - Por exemplo, se um inteiro da entrada está entre 0 e $3x10^9$, o tipo *int* não servirá;
  - Consequentemente, um programa que usa o tipo *int* funcionará para alguns casos e outros não.

Problemas *Ad Hoc*

Problemas *Ad Hoc*

- *Ad Hoc*
  - Para isto, para um determinado ato;
  - **Para este caso específico**;
  - Eventualmente investido em função provisória, para um fim especial.

Problemas *Ad Hoc*

- Problemas *Ad Hoc* são aqueles cujos algoritmos de solução não recaem em categorias bem estudadas;
- Cada problema é diferente do outro
  - Não existe técnica específica ou genérica para resolvê-los.
- São os mais divertidos (às vezes frustrantes)
  - Desafios novos a cada problema.
- A solução pode exigir uma estrutura nova ou um conjunto nada usual de laços e condições.

Problemas *Ad Hoc*

- Às vezes requerem combinações especiais raras, ou pelo menos, raramente encontradas;
- Exigem leitura cuidadosa do enunciado
  - A solução pode ser o ataque a cada uma das dicas dadas no enunciado.
- Por serem tão livres, é necessário vigiar-se para não cair em códigos ineficientes
  - Por exemplo, cinco laços aninhados…

Problemas *Ad Hoc*

- Problemas *Ad Hoc* podem ser superficialmente categorizados
  - Triviais, fáceis, médios...
  - Jogos (cartas, tabuleiros, etc.);
  - Combinatórios (Josephus, Palíndromos, Anagramas);
  - Problemas reais/Simulação;
  - Simplesmente "*Ad Hoc*".

Problemas *Ad Hoc*

Truque de Cartas

- Um mágico faz o seguinte truque tendo *n* cartas do espadas com a face para baixo:
  - Se retirar a carta do topo, a próxima será o ás;
  - Se retirar duas cartas, a próxima será o 2;
  - Se retirar três cartas, a próxima será o 3;
  - …
  - Cartas retiradas voltam ao fundo das *n* cartas, menos as viradas.
- Qual é o algoritmo de embaralhamento das cartas?

Problemas Selecionados

- http://br.spoj.com/problems/QUADRAD2/
- http://br.spoj.com/problems/FATORIA2/
- http://br.spoj.com/problems/QUERM/
- http://br.spoj.com/problems/JSEDEX/
- http://br.spoj.com/problems/OVERF09/

Perguntas?