UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
UTFPR

APOSTILA DA OLIMPÍADA BRASILEIRA DE INFORMÁTICA

Fevereiro – 2014
CURITIBA - PR

0 Introdução

A Olimpíada Brasileira de Informática é uma competição voltada a despertar o interesse dos competidores para a computação. Através de exercícios práticos e interessantes o aluno aprende a programar e conhece melhor o dia a dia de um profissional de computação.

O objetivo desta apostila é apresentar o necessário para realizar a prova da Olimpíada Brasileira de Informática (OBI) para alunos do ensino técnico. Comparada a uma apostila de um curso comum de programação, ela não apresentará todos os tipos de dados e nem detalhes de interface com o usuário, porém mostrará o suficiente de programação, estrutura de dados e grafos para resolver os problemas da olimpíada.

A linguagem de programação é a forma mais completa de passar instruções ao computador. Através de várias instruções simples são feitos programas complexos. Na olimpíada é permitido o uso de cinco linguagens: Pascal, C, C++, Python e Java. Os códigos desta apostila estão em C++, mas podem ser facilmente convertidos para as outras linguagens de programação.

O treinamento para OBI repete-se todo o ano e se perder alguma aula pode-se pedir ajuda à um dos professores nas aulas ou na maratona de programação. Além disso as aulas concentram-se principalmente em atividades práticas e podem ser facilmente puladas sem perda de continuidade do conteúdo. Portanto, concentre mais nas suas outras disciplinas, pois elas podem te atrasar um semestre.

Sumário

1 Regras OBI

Todas as informações oficiais sobre a olimpíada estão disponíveis no site [1], aqui tem um resumo de todas as regras e informações do site no ano de 2013.

1.1 Organização em Modalidades

A competição é dividida em modalidades destinadas a diversas idades/formações do aluno. A tabela 1 resume as modalidades.

Tabela 1: Relação da modalidade com o ano do curso do aluno.

Nome Modalidade	Curso Ofertado	Público Alvo
Iniciação Nível 1	Não	Até 7º ano do Ensino Fundamental
Iniciação Nível 1	Não	Até 9º ano do Ensino Fundamental
Programação Júnior	Não	Até 9º ano do Ensino Fundamental
Programação Nível 1	Sim	Até o 2º ano do Ensino Médio/Técnico
Programação Nível 2	Sim	Até o 4º ano do Ensino Médio/Técnico ou 1º ano da Graduação

As modalidades de iniciação são compostas de problemas de lógica e computação, sem programação ou o uso de computador, a resolução destes requer apenas o uso de lápis e papel. O site da olimpíada contém todas as provas e seus gabaritos comentados. O PET-ECO não oferece treinamento para essa modalidade.

Programação Júnior é uma outra modalidade que não oferecemos curso. Esta modalidade é recente e não possui muitos problemas dificultando a elaboração da apostila além de que seria necessário uma apostila apenas para essa modalidade.

Esta apostila é destinada as modalidades Programação Nível 1 e 2. São elegíveis a estas provas alunos até o último ano do ensino médio/técnico e que estejam no primeiro ano da faculdade tendo terminado o ensino médio no ano anterior.

A parte teórica inicial do começo da apostila é igual, os capítulos finais são destinados apenas ao nível 2. Os problemas são dirigidos para cada nível, no cabeçalho dos problemas é indicado em qual prova o problema caiu. O site da olimpíada fornece as provas passadas em [3] e um sistema para teste automático de problemas [4].

[1] (http://olimpiada.ic.unicamp.br/)
[2] (http://olimpiada.ic.unicamp.br/info_geral/regulamento)
[3] (http://olimpiada.ic.unicamp.br/passadas/pp)
[4] (http://olimpiada.ic.unicamp.br/pratique/programacao)

1.2 Datas das Provas e Inscrição

Ainda não foram definidas as datas para o ano de 2014, mas a tabela 2 representa as datas do ano de 2013.

Tabela 2. Fonte: Retirado de [http://olimpiada.ic.unicamp.br/info_geral/datas]

	Fase 1		Fase 2	
Modalidade	**Dia**	**hora**	**Dia**	**hora**
Programação Nível 2	09/03/2013	13:00-18:00	06/04/2013	13:00-18:00
Programação Nível 1	18/05/2013	13:00-17:00	31/08/2013	13:00-17:00
Programação Nível Júnior		09:00-12:00		09:00-12:00
Iniciação Nível 1		09:00-11:00		09:00-11:00
Iniciação Nível 2		09:00-11:00		09:00-11:00

A inscrição é feita no dia da olimpíada.

1.3 Premiação

Todos os participantes receberão certificados de participação nas provas. Os melhores colocados receberão medalhas de ouro, prata e bronze.

Os melhores classificados nas modalidades de programação são convidados para um curso na UNICAMP de Aperfeiçoamento em Programação. Os selecionados recebem custeio para irem fazer o curso. Ao final do curso os melhores alunos participam da Olimpíada Internacional de Informática (IOI – International Olympiad in Informatics).

1.4 Linguagens de Programação

Na Olimpíada Brasileira de Informática são permitidas as seguintes linguagens de programação:

- Pascal
- C
- C++
- Python
- Java

Durante as aulas e a divulgação das soluções no site da olimpíada usa-se principalmente a linguagem C++, mas o aluno está livre para utilizar qualquer linguagem durante a aula. É ofertado um curso de Python pelo PET-COCE.

Já na Olimpíada Internacional de Informática são permitidas as seguintes linguagens de programação:

- C
- C++
- Pascal

2 Montando seu Ambiente de Trabalho

Este capítulo apresenta os conceitos iniciais sobre a linguagem C++.

2.1 História

Na década de 1970 foi criada a linguagem C. Esta linguagem foi usada para desenvolver o sistema operacional UNIX versão 5. A popularidade dos microcomputadores fez surgir diversas implementações do C, no entanto, muitas vezes, um programa escrito para um sistema não rodava no outro sem modificações. A padronização surgiu em 1983 estabelecido pelo ANSI (American National Standards Institute).

O C++ foi desenvolvido como uma extensão do C por Bjarne Stroustrup no início da década de 80 no Bell Laboratories. C++ incluiu várias melhorias ao C como a possibilidade de um código mais limpo e a orientação a objetos, além de ainda incluir os elementos de baixo nível do C. A padronização surgiu em 1998 como uma norma ISO.

2.2 Escolha do C++

Um dos objetivos da olimpíada é despertar o interesse do aluno à computação para que ele use programação além da olimpíada, como em um curso técnico, de graduação ou de microcontroladores, e o C++ é mais abrangente nestas áreas.

Tradicionalmente é iniciado o aprendizado da linguagem de programação C que não possui algumas funções do C++. No entanto, no curso é ensino os tópicos que são tradicionalmente abordados em cursos iniciantes. Além do mais o compilador do C++ aceita construções de código que o de C não aceita.

Na UTFPR Curitiba é ofertado, pelo PETECO e pelo PETCOCE, o curso de Arduíno, um microcontrolador programado em uma linguagem baseada no C++.

A inclusão das linguagens Python e Java é recente na OBI e por isso a divulgação de possíveis soluções dos problemas da OBI foi e continua sendo feita principalmente na linguagem C++.

2.3 Ambiente de Programação

O ambiente utilizado para programação é o Code::Blocks. Para realizar os exercícios em casa baixe-o em http://sourceforge.net/projects/codeblocks/files/Binaries/10.05/Windows/codeblocks-10.05mingw-setup.exe (windows) ou http://sourceforge.net/projects/codeblocks/files/Binaries/10.05/MacOS/codeblocks-10.05-p1-mac.dmg (Mac OS) e instale.

Na primeira vez que rodar o Code::Blocks será necessário escolher um "*Compiler*", escolha um que tenha sido detectado, clique em "Set as default" e depois em Ok. A figura 2.1 mostra que pergunta sobre o compilador.

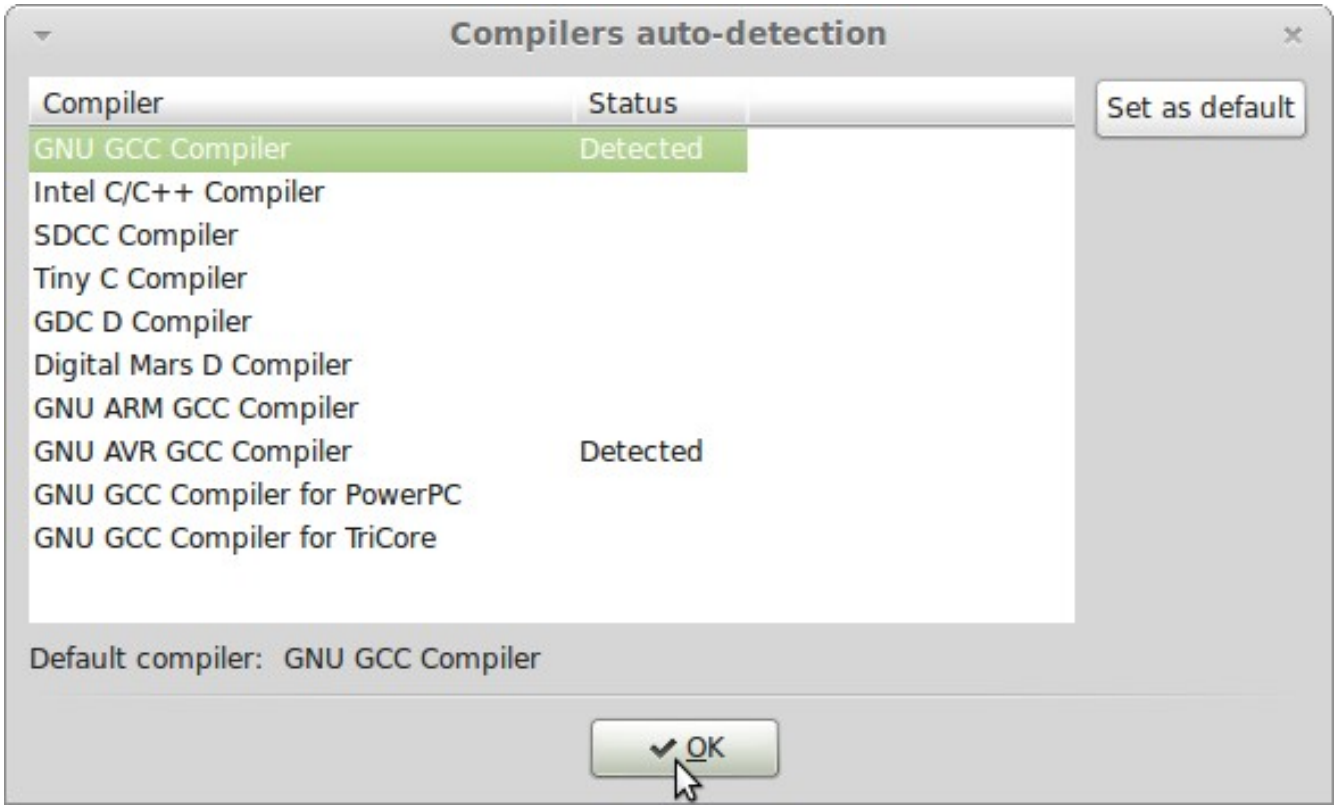


Figura 2.1: Escolha do compilador na primeira execução do Code::Blocks.

2.4 Primeiro Programa

Com o Code::Blocks aberto crie um novo projeto (File->New->Project...), escolha um projeto de “Console application”, clique no botão Go, e duas vezes em Next, escolha um título para o projeto e uma pasta para armazená-lo.

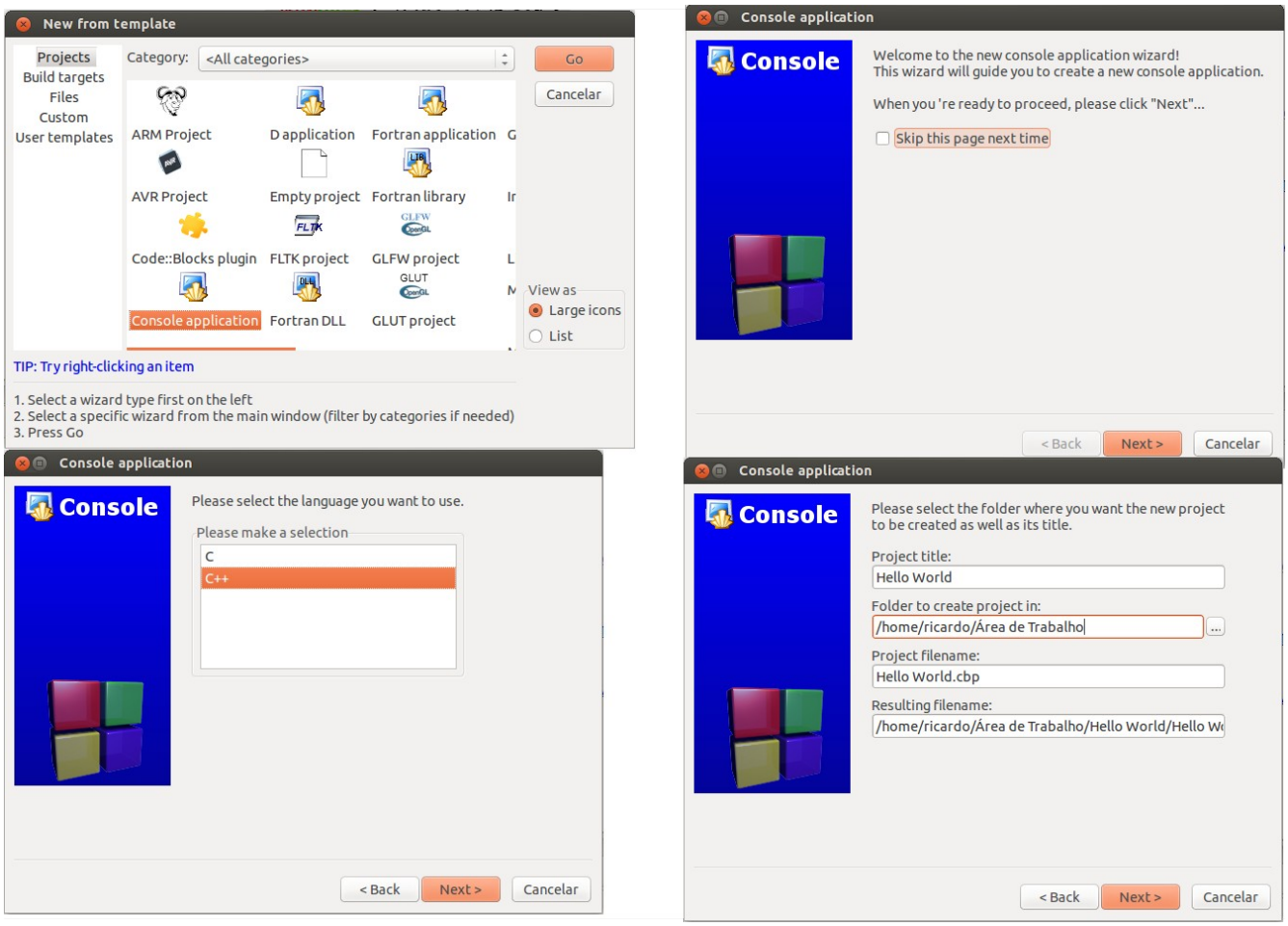


Figura 2.2: As quatro etapas para criar um novo programa em C++: primeiro escolhe-se o tipo de aplicação (Console Application), em seguida lê-se a página de bem vindo, no terceiro passo escolhe-se a linguagem C++, por fim preenche-se o nome do projeto e escolhe-se uma pasta para salvá-lo.

Em seguida abra no diretório do programa a esquerda o projeto criado, e então a pastas Sources e por fim o código main.cpp. A figura 2.3 mostra o resultado obtido.

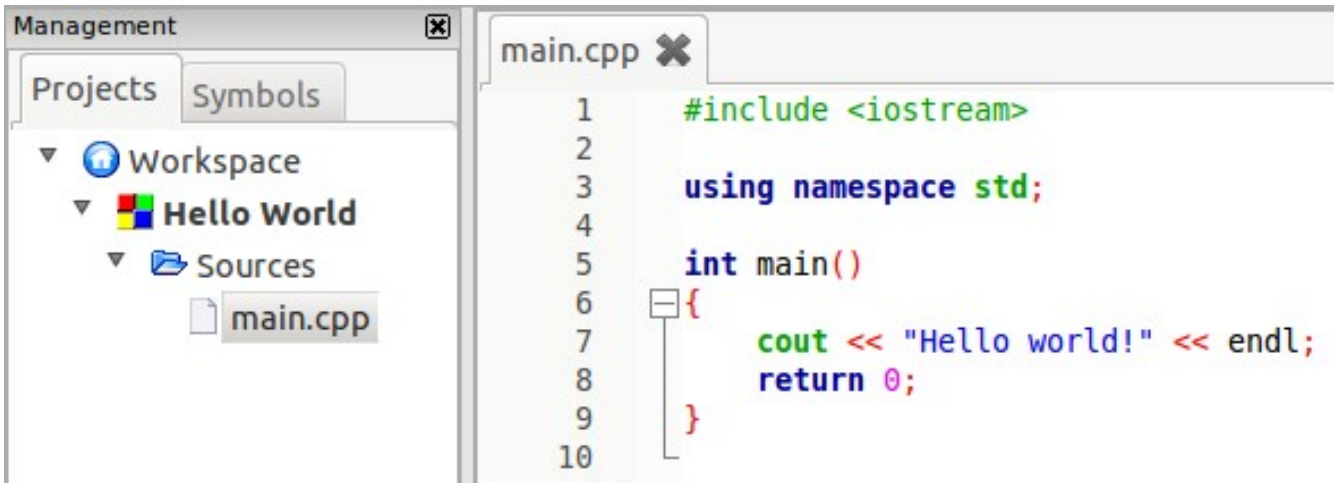


Figura 2.3: O caminho para abrir o arquivo main.cpp e o código inicial do programa.

Texto exibido na parte direita da figura 2.3 mostra o código de um programa simples. As três primeiras linhas são cabeçalhos necessários para mostrar na tela mensagens, o tópico avançado

deste capítulo explica melhor a terceira linha. A linha 5 declara a função main (principal) que é o ponto de partida do programa, é possível ter mais funções como será mostrado no capítulo 5. A linha 7 é o comando para escrever na tela a mensagem entre aspas, essa mensagem pode ser qualquer frase, o "endl" indica o final de uma linha. O "return 0" será abordado no capítulo 5 quando será discutido as funções.

2.5 Avançado: using namespace std

A principal diferença entre o C e o C++ é que o C++ suporta objetos. O paradigma orientação a objetos pode ser visto como um modo de organizar o código e fonte do programa afim de obter a reutilização de código. Para utilizar trechos prontos de outro arquivo é necessário usar a diretiva #import seguida do nome do arquivo. Em orientação a objetos as funções são organizadas dentro de classes e um jeito simples de utilizá-las é escrever o nome da classe, seguido de "::" e por fim com o nome da função desejada. Deste modo para chamar a função cout é necessário escrever std::cout, mas como digitar "std::" é uma tarefa cansativa e repetitiva basta adicionar no começo do programa a linha "using namespace std;" que é reconhecido todas as funções da classe std sem a necessidade de digitá-lo.

3 Variáveis e Controle de Fluxo

A olimpíada de informática exige muito raciocínio lógico e pouco conhecimento decorado das funções da linguagem de programação. Seguindo esse modelo são apresentados apenas os tipos básicos de dados e a estrutura de controle if visando já começar a resolver problemas reais da olimpíada de informática.

3.1 Variáveis

Dentro de um programa é necessário guardar dados adquiridos para serem processados posteriormente. Usa-se variáveis para nomear os dados e usá-los. Os tipos de dados mais fáceis de processar são os números inteiros e reais. Acompanhe o exemplo do código 1:

```
1  #include <iostream>
2  using namespace std;
3
4  int main(){
5      int peso;
6      float altura, IMC;//Índice de Massa Corporal
7      peso = 60;
8      altura = 1.80;
9      IMC = peso/(altura*altura);
10     cout << "O Indice de Massa Corporal calculado é de " << IMC << endl;
11     return 0;
12 }
```

Código 1: Calculo do índice de massa corporal baseado no peso e na altura.

Na linha 5 é criada uma variável inteira com o nome peso e na linha 6 são criadas duas variáveis numéricas com números após a vírgula (float point) com os nomes altura e IMC. A sintaxe[1] básica para criar uma variável é o tipo da variável, um espaço e o nome da variável procedido de um ponto e vírgula, ainda é possível criar várias variáveis em uma mesma expressão separando por vírgula ou atribuir valores iniciais as variáveis em sua criação. O nome de uma variável deve começar por uma letra ou sublinhado (_) e possuir apenas letras, números e sublinhados.

As linhas 7, 8 e 9 contém atribuições. O operador igual (=) atribui o valor da expressão matemática de sua direita à variável a sua esquerda. A linha 9 calcula a expressão $\frac{peso}{altura^2}$, pela falta de um operador aritmético de exponenciação a variável altura foi multiplicada por ela mesma. A ordem da realização das operações aritméticas é a convencional, primeiro realiza-se multiplicações e divisões e depois somas e subtrações, quando acontecerem duas operações com a mesma precedência executa-se a mais à esquerda.

A tabela 1 apresenta uma lista com os principais operadores necessários na prova. É necessário saber usar os 5 primeiros operadores da tabela, já as outras funções matemáticas tipicamente encontradas em calculadoras científicas estão enunciadas na página do manual da biblioteca math.h. Para acessar esse manual abra um terminal no Linux e digite "man math.h".

[1] Sintaxe é o formato de uma expressão.

Tabela 1: Principais operadores e funções usados na olimpíada.

Operador	Comentário	Exemplo
+	Soma	int a = 3 + 2;//a será 5
-	Subtração	int a = 3 – 2;//a será 1
*	Multiplicação	int a = 3*2;//a será 6

/	Divisão	int a = 6/2;//a será 3
		int a = 5/2;//a será 2
		float a = 5/2;//a será 2
		float a = 5.0/2;//a será 2,5

%	Resto da divisão inteira	int a = 5%2;//a será 1
pow*	Exponencial	float a = pow(3, 2);//a será 9
sqrt*	Raiz quadrada	float a = sqrt(2);//a será 1,41421

3.2 Controle de Fluxo Condicional

A estrutura de controle if é usada para fazer o computador realizar diferentes operações dependendo de uma expressão lógica. O código 2 continua o exemplo anterior e mostrando um mensagem personalizada para o usuário dependendo do seu IMC.

```
1  #include <iostream>
2  using namespace std;
3
4  int main(){
5      int peso;
6      float altura, IMC;//Índice de Massa Corporal
7      peso = 60;
8      altura = 1.80;
9      IMC = peso/(altura*altura);
10     cout << "O Indice de Massa Corporal calculado é de " << IMC << endl;
11
12     if( IMC < 18.5){
13         cout << "Magro" << endl;
14     }
15     if( IMC >= 18.5 && IMC < 25 ){
16         cout << "Saudável" << endl;
17     }
18     if( IMC >= 25){
19         cout << "Obeso" << endl;
20     }
21     return 0;
22 }
```

Código 2: Atualização do código 1 para imprimir uma mensagem personalizada para cada faixa de peso.

Considere o problema de determinar se um aluno deve ser aprovado ou não considerando a nota e a frequência na UTFPR, a imagem1 ilustra o diagrama em blocos, enquanto o código 3 demonstra a solução implementada em uma linguagem de programação.

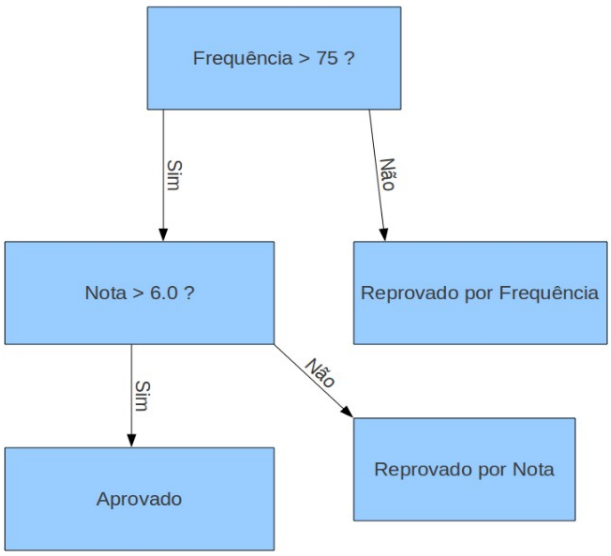


Figura 3.1: Diagrama em blocos do sistema de aprovação da UTFPR.

```
1  #include <iostream>
2  using namespace std;
3
4  int main(){
5      float nota, frequencia;
6      cout << "Digite a nota e a frequencia ";
7      cin >> nota;
8      cin >> frequencia;
9      if( frequencia < 75 ){
10         cout << "Reprovado por frequencia" << endl;
11     }else{
12         if( nota > 6.0){
13             cout << "Aprovado" << endl;
14         }else{
15             cout << "Reprovado por nota" << endl;
16         }
17     }
18     return 0;
19 }
```

Código 3: Código verificando se um aluno está aprovado segundo as diretrizes da UTFPR.

3.3 Avançado: Mais Tipos de Dados

Os tipos de dados necessários para estar pronto para qualquer problema da OBI estão na tabela 3.1. É importante conhecer os limites de cada variável, pois, às vezes, o enunciado de algum problema indica o tamanho máximo usado e em geral ele é menor do disponível nos tipos de

variáveis comuns.

Tabela 3.1: Tipos de dados comuns na OBI.

Tipo	Faixa de valores	Descrição
char	-127 a 128	Representa um carácter
int	-2.147.483.647 a 2.147.483.648	Representa um inteiro
long	$-9x10^{18}$ a $9x10^{18}$	Representa um inteiro
float	-10^{37} a 10^{37} com seis casas de precisão	Representa um número real
double	-10^{308} a 10^{308} com quinze casas decimais	Representa um número real
void		Sem valor

3.4 Avançado: printf e scanf

As funções printf e scanf permitem ler e escrever no console com algumas opções de formatação a mais que o cout e cin, porém com o uso mais complicado. A função printf escreve dados no console enquanto que a função scanf lê dados do console. Para usa-las é necessário especificar o formato do dado a ser lido ou escrito e as variáveis que serão usadas. Um exemplo do uso do printf para exibir o valor de variáveis está no código 4.

```
int a = 2;
float b = 5.5;
printf("a vale %d e b vale %f", a, b);
//imprime: a vale 2 e b vale 5.5
printf("a vale %3d e b vale %.3f", a, b);
//imprime: a vale 002 e b vale 5.500
```

Código 4: Exemplos do uso do scanf.

Neste código é importante notar que o símbolo %d é substituído pelo inteiro a e o símbolo %f é substituído pela variável real b na escrita no console. Cada tipo de variável deve ser substituída por um símbolo diferente, a tabela 2.1 mostra o código de cada formato de dado à ser usado no printf e no scanf. Ainda no código de exemplo 4 são colocados números para alterar o formato de exibição da variável.

Tabela 2.1: Códigos usados pelo scanf e printf para processar variáveis.

Código	Formato
%c	Caractere (char)
%s	String (char[])
%d	Inteiro (int)
%ld	Long (long)
%f	Real (float)
%lf	Real (double)

O código 4 apresenta um exemplo de uso do scanf para leitura de um inteiro. Observe a presença de um "&" antes da variável que guardará o valor lido, este símbolo é necessário na leitura dos formatos apresentados menos do formato string, por hora não é possível explicar o motivo da necessidade deste símbolo.

```
scanf("%d", &x);
```

Código 5: leitura de um número inteiro para a variáveil x

3.5 Problemas

Nível 1 Fase 1
OBI http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2008f1p1_obi
Aviões de Papel http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2009f1p1_papel
Overflow http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2009f1p1_overflow
Triângulos http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2011f1p1_triangulos
Vice-campeão http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2012f1p1_vice
Detectando Colisões
http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2007f1p1_colisoes
Nível 1 Fase 2
Frota de Taxi
Sedex Marciano http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2010f2p1_marciano
Nível 2 Fase 1
Avião http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2008f1p2_aviao
Cometa http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2010f1p2_cometa
Nível 2 Fase 2
Notas da Prova http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2009f1p2_nota
Álbum de fotos http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2012f2p2_album

4 Estruturas de Repetição

Neste capítulo é apresentado as estruturas de repetição for e while. Em muitos problemas elas são usadas simplesmente para ler a entrada e/ou repetir várias vezes a resolução do problema. Além destas estruturas de repetição é introduzida uma estrutura de dados para armazenar vários valores do mesmo formato, os vetores.

4.1 Estruturas de Repetição

Neste capítulo usaremos o for de forma simples para repetir um comando uma quantidade determinada de vezes. A forma genérica do comando for é:

```
for (inicialização; condição; incremento){
	//comandos
}
```

O comando de inicialização é executado apenas uma vez ao iniciar o comando for. A condição é testada sempre no início das repetições e se for falsa encerra a execução do for. E o incremento é executado no final de cada execução. Os comandos entre as chaves são executados diversas vezes até que a condição deixe de ser verdadeira. O código 4.1 executa o comando cout contando de 0 a 9.

```
for(int i = 0; i < 10; i++){
	cout << "O valor contado é " << i << endl;
}
```

Percorrer valores ou executar alguma sequência de comandos diversas vezes são os usos mais comuns do for. Se focando nesses casos comuns é fácil usar o for.

Uma versão mais simples do for é o while. Este comando é equivalente a um for sem um comando de inicialização e sem um comando de incremento. Sua sintaxe é:

```
while(condição){
	//comandos
}
```

O while é geralmente usado na olimpíada para repetir comandos até que uma entrada específica do programa seja fornecida ou quando não é sabido desde o início quantas vezes a repetição deve acontecer.

4.2 Vetores

Para guardar muitas variáveis do mesmo tipo e numerá-las existem os vetores. Muitas vezes é necessário guardar um número potencialmente grande de variáveis e ler da entrada seus valores. O código exemplifica o uso de um vetor para mostrar o valor do fibonatti.

```cpp
#include <iostream>
using namespace std;

int main(){//imprime os 10 primeiros números da sequência de Fibonatti
    int fib[10];
    fib[0] = 1;
    fib[1] = 1;
    for(int i = 2; i < 10; i++){
        fib[i] = fib[i-1] + fib[i-2];
    }
    for(int i = 0; i < 10; i++){
        cout << fib[i] << " ";
    }
}
```

Código 4.1: exemplo para mostrar na tela os dez primeiros números da sequência de Fibonatti.

4.3 Avançado: break

Às vezes é conveniente sair de um laço de repetição sem esperar o passo da verificação ou quando alguma condição específica é atingida durante o laço. Nestes casos existe o comando break para fazer o programa sair do loop, veja o exemplo que lê uma entrada e sai do loop se a entrada for 0:

```cpp
#include <iostream>
using namespace std;

int main(){
    int n;
    while(true){
        cin >> n;
        if(n == 0){
            break;
        }
        //comandos
    }
}
```

Código 4.2: exemplo do uso do break.

4.4 Problemas

Nível 1 Fase 1
Quadrado Mágico http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2007f1p1_magico
Conta de água http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2010f1p1_conta
Pedágio http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2010f1p1_pedagio
Progressões Aritméticas http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2011f1p1_pas
Consecutivos http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2012f1p1_iguais
Nível 1 Fase 2
Maratona http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2009f2p1_maratona
Balé http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2011f2p1_bale
Selos http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2011f2p1_selos
Nível 2 Fase 1
Ogros http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2008f1p2_ogros
O Fugitivo http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2009f1p2_fugitivo
Caçadores de Mitos http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2009f1p2_mito

Elevador http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2010f1p2_elevador
Nível 2 Fase 2
Telemarketing http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2007f2p2_tele
Soma das casas http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2012f2p2_soma

5 Letras e funções

Este capítulo aprimora diversos conceitos apresentados: um novo tipo de dado é mostrado, funções que são usadas principalmente para melhorar a legibilidade de seu código, matrizes que são vetores de duas dimensões e as páginas do manual que permitem consultar listas de funções e exemplos de código úteis.

5.1 O tipo de dado Carácter (char)

O computador processa letras pelo formato de dado char. Internamente o computador representa cada letra ou símbolo da tabela ASCII por um número, para facilitar a programação existem meios para tratá-los apenas com os símbolos facilmente legíveis por humanos. Para fins de curiosidade o código 5.1 imprime a tabela ASCII.

```
1  #include <iostream>
2  using namespace std;
3
4  int main(){
5      for(int i = 0; i < 128; i++){
6          cout << "Código\tSimbolo" << endl;
7          cout << i << "\t" << (char)i << endl;
8      }
9  }
```

Código 5.1: Código que monta o código ASCII. O "\t" é um TAB e o (char) converte do código numérico para a representação simbólica.

O código 5.2 exemplifica o uso do char resolvendo o problema Telefone. Observe que é lido um vetor inteiro de letras na linha 9. O laço de repetição for percorre o vetor até encontrar o carácter que indica o final de um array de char's, o '\0'. Os if's processam a palavra lida como números, comparando o valor numérico de cada letra com o valor numérico dos caracteres -, 0 e 9. Por fim, o vetor conversor associou cada letra a um número.

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(){
6                      //{A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y}
7      int conversor[] = {2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 9, 9, 9};
8      char numero[16];
9      cin >> numero;
10     for(int i = 0; numero[i] != '\0'; i++){
11         if(numero[i] == '-')
12             cout << "-";
13         else if(numero[i] >= '0' && numero[i] <= '9')
14             cout << numero[i];
15         else
16             cout << conversor[numero[i]-'A'];
17     }
18 }
```

Código 5.2: Resolução do problema do Telefone de 2008 nível 1 fase 1.

5.2 Funções

Aqui as funções servirão para deixar mais fácil de ler o código. As funções podem retornar um valor, como uma função matemática, ou apenas realizar operações. O código exemplifica a criação de uma função para testar se um número é primo e retorna um valor que significa simbolizando o resultado. A escolha dos números 0 e 1 como retorno foi arbitrária.

```
1  #include <iostream>
2  using namespace std;
3
4  int ehPrimo(int n){
5      for(int i = 2; i < n; i++){
6          if(n % i == 0){
7              return 0;//número digitado não é primo
8          }
9      }
10     return 1;//número digitado é primo
11 }
12
13 int main(){//programa verifica se número digitado é primo
14     int n;
15     cin >> n;
16     if(ehPrimo(n) == 1){
17         cout << "O número digitado é primo" << endl;
18     }else{
19         cout << "O número digitado não é primo" << endl;
20     }
21 }
```

Código 5.3: Exemplo da criação de uma função que determina se um número é primo.

Uma segunda utilidade das funções é oferecer rotinas prontas para tarefas comuns. Como apresentado no capítulo 3, existem as funções matemáticas que estão listadas no manual do math.h. Um trecho de código muito importante é o do quicksort, um algoritmo de ordenação. No apêndice existe uma lista com os diversos manuais, veja o item 13.1 qsort.

5.3 Matrizes

Já foram apresentadas as listas de dados (vetores), imagine agora a necessidade de tabelas de dados que podem representar mapas, distâncias ou ligações entre listas. A criação é simples, no entanto a manipulação de matrizes exige muitas vezes um laço (for) dentro de outro. O código 5.4 exemplifica a leitura das dimensões de uma matriz, sua alocação em memória, a leitura de seus valores e a escrita na tela de seus valores. É comum em na manipulação de matrizes ou chamadas de funções que trabalhem com matrizes nomear a variável que indica com a linha de l e a que indica com a coluna de c.

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Insira o tamanho da matriz a ser criada" << endl;
    int m, n;
    cin >> m;//número de linhas
    cin >> n;//número de colunas
    int matriz[m][n];//cria matriz de tamanho MxN
    for(int i = 0; i < m; i++){
        for(int j = 0; j < n; j++){
            cin >> matriz[i][j];//lê e coloca na posição (i,j)
        }
    }
    for(int l = 0; l < m; l++){//l é a linha
        for(int c = 0; c < n; c++){//c é a coluna
            cout << matriz[l][c] << " ";//escreve valor em (l,c)
        }
        cout << endl;
    }
    return 0;
}
```

Código 5.4: Código de exemplo para mexer com matrizes.

5.4 Avançado: string's e funções de manipulação de string's

String é uma cadeia de caracteres, em muitos casos são palavras. Nos primeiros anos da OBI alguns problemas exigiam o conhecimento do uso de strings, mas nos últimos 5 anos nenhum problema exige esse conhecimento. De qualquer forma para programar é bem útil saber manipular palavras entradas pelo usuário.

Como o nome sugere, string's são cadeias (sequências) de caracteres. Para representá-los é usado um vetor do tipo char. O final das string's é indicado com o símbolo de código 0, por isso é sempre necessário colocar um espaço a mais na string. A leitura do console coloca este símbolo automaticamente.

As funções de manipulação de string estão listadas no manual string.h, as principais são: strcmp que compara duas strings e informa qual vem antes em ordem da tabela ASCII; e strlen que retorna o tamanho de uma string.

5.5 Problemas

Nível 1 Fase 1
Telefone http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2008f1p1_telefone
Peça Perdida http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2007f1p1_perdida
Corrida http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2011f1p1_corrida
Nível 1 Fase 2
Telefone http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2008f1p2_telefone
Lista de Chamada http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2010f2p1_chamada
Nível 2 Fase 1
Batalha Naval http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2010f1p2_batalha

Caça ao Tesouro http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2011f1p2_tesouro
O Mar não está Para Peixe
http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2011f1p2_pesca
Nível 2 Fase 2
Olímpiada http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2009f2p2_olimpiada
Pré 2005

Campo de Minhocas

Arquivo fonte: minhoca.c, minhoca.cc, minhoca.cpp ou minhoca.pas

Minhocas são muito importantes para a agricultura e como insumo para produção de ração animal. A Organização para Bioengenharia de Minhocas (OBM) é uma entidade não governamental que promove o aumento da produção, utilização e exportação de minhocas.

Uma das atividades promovidas pela OBM é a manutenção de uma fazenda experimental para pesquisa de novas tecnologias de criação de minhocas. Na fazenda, a área destinada às pesquisas é de formato retangular, dividida em células quadrangulares de mesmo tamanho. As células são utilizadas para testar os efeitos, na produção de minhocas, de variações de espécies de minhocas, tipos de terra, de adubo, de tratamento, etc. Os pesquisadores da OBM mantêm um acompanhamento constante do desenvolvimento das minhocas em cada célula, e têm uma estimativa extremamente precisa da produtividade em cada uma das células. A figura abaixo mostra um mapa da fazenda, mostrando a produtividade estimada de cada uma das células.

81	28	240	10
40	10	100	240
20	180	110	35

Um pesquisador da OBM inventou e construiu uma máquina colhedeira de minhocas, e quer testá-la na fazenda. A máquina tem a largura de uma célula, e em uma passada pelo terreno de uma célula colhe todas as minhocas dessa célula, separando-as, limpando-as e empacotando-as. Ou seja, a máquina eliminará uma das etapas mais intensivas de mão de obra no processo de produção de minhocas. A máquina, porém, ainda está em desenvolvimento e tem uma restrição: não faz curvas, podendo movimentar-se somente em linha reta.

Decidiu-se então que seria efetuado um teste com a máquina, de forma a colher o maior número possível de minhocas em uma única passada, em linha reta, de lado a lado do campo de minhocas. Ou seja, a máquina deve colher todas as minhocas de uma 'coluna' ou de uma 'linha' de células do campo de minhocas (a linha ou coluna cuja soma das produtividades esperadas das células é a maior possível).

Tarefa

Escreva um programa que, fornecido o mapa do campo de minhocas, descrevendo a produtividade estimada em cada célula, calcule o número esperado total de minhocas a serem colhidas pela máquina durante o teste, conforme descrito acima.

Entrada

A primeira linha da entrada contém dois números inteiros N e M, representando respectivamente o número de linhas ($1 \leq N \leq 100$) e o número de colunas ($1 \leq M \leq 100$) de células existentes no campo experimental de minhocas. Cada uma das N linhas seguintes contém M inteiros, representando as produtividades estimadas das células correspondentes a uma linha do campo de minhocas.

A entrada deve ser lida do dispositivo de entrada padrão (normalmente o teclado).

Saída

A saída deve ser composta por uma única linha contendo um inteiro, indicando o número esperado total de minhocas a serem colhidas pela máquina durante o teste.

A saída deve ser escrita no dispositivo de saída padrão (normalmente a tela).

Restrições

$1 \leq N \leq 100$
$1 \leq M \leq 100$
$0 \leq$ *Produtividade de uma célula* ≤ 500
$0 \leq$ *Produtividade de uma linha ou coluna de células* ≤ 50000

Exemplos de entrada e saída

Exemplo 1

Entrada	**Saída**
3 4 81 28 240 10 40 10 100 240 20 180 110 35	450

Exemplo 2

Entrada	**Saída**
4 1 100 110 0 100	310

6 Novos Formatos de Dados e Revisão

Os formatos de dados importantes para a OBI já foram apresentados, mas se tratando de uma olimpíada, sempre é bom saber um pouco mais, pois pode aparecer um problema mais exigente que o comum.

6.1 Formatos de dados Long e Double

Os formatos long e double correspondem, respectivamente, aos formatos int e float, porém com mais espaço de dados. A tabela a seguir mostra os formatos de dados já apresentados e também a capacidade de armazenar dados de cada um.

Tabela 6.1: Comparação do tamanho do dado entre os tipos de dados previamente apresentados e os novos.

Nome	Alcance em potência	Alcance
int	-2^{31} até 2^{31} -1	2.147.483.648 até 2.147.483.647
long	-2^{63} até 2^{63} -1	9.223.372.036.854.775.808 até 9.223.372.036.854.775.807
float		
double		
char	0 até 2^{8}-1	0 até 255
char[]		

Não é necessário decorar o tamanho dos dados, eles estão listados aqui por dois motivos. O primeiro e mais importante é que às vezes no enunciado de alguns problemas é apresentado os limites de tamanho dos resultados em potência de dois ou em números longos, para o aluno não se assustar é necessário saber dos limites e verificar que tudo ocorrerá bem. O segundo é a necessidade de usar algumas bibliotecas do sistema que retornam ou exigem o uso de algum destes novos dados.

Uma coisa importante de saber é converter dados do tipo real para o tipo inteiro e principalmente vice versa. Nas atribuições de variáveis ele são feitos automaticamente, mas no meio da conta eles são convertidos para o formato do operador à esquerda. Veja o exemplo a seguir, aonde a divisão de um inteiro por um número real resulta em um inteiro e provavelmente era desejado um número real.

[TODO] fazer exemplo de divisão não desejada.

6.2 Agrupando Variáveis em uma Só: Struct

Para aumentar a legibilidade do código muitas vezes é interessante agrupar variáveis com um nome mais geral e ter um subnome para cada variável deste grupo. O exemplo a seguir usa este tipo de estrutura de dados para facilitar lembrar os nomes de variável durante programação e a entender o programa depois de pronto.

//encontrar problema que use structs

Avançado: Mais Erros no Compilador

É possível pedir para o compilador exibir mais mensagens de erros e possíveis erros, para isto deve-se adicionar a opção allWarnings que é abreviada como -Wall, para isto abra no menu Project, clique em Build options... e na tela que abrir marque o checkbox [-Wall]. Pronto, a partir deste momento o compilador verificará por mais erros durante a compilação ajudando a corrigir

erros mais difíceis de serem encontrados.
Novos Format
Novos tipos de dados (long, double, string), cast, for duplo
avançado todos os formatos de dados
avançado mais erros no compilador
Nível 1 Fase 1
Quadrado Mágico http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2007f1p1_magico
Pulo do Sapo http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2011f1p1_pulosapo
Nível 1 Fase 2
Balé http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2011f2p1_bale
Nível 2 Fase 1
Repositórios http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2007f1p2_repos
Nível 2 Fase 2
Soma das Casas http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2012f2p2_soma

7 Quicksort

Neste capítulo pretende-se ensinar o uso da função quicksort, sem mostrar ou ao menos passar o funcionamento intuitivo interno desta função.

7.1 man qsort

O comando "man qsort" quando digitado em um terminal de algum sistema Linux mostra a página de manual da função quicksort com um exemplo de uso. Recomendamos adaptar este exemplo quando for necessário ordenar dados. O anexo13.1 é uma cópia deste manual, código 7.1 é o exemplo de código do manual que deve ser adaptado. O exemplo.

Anterior a 2005
Sorvete 2001
Nível 1 Fase 1
Times http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2010f1p1_times
Nível 1 Fase 2
Olimpíadas http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2009f2p1_olimp
Nível 2 Fase 1
Pedido de Desculpas 2005

Pedido de Desculpas

Arquivo fonte: desculpa.c, desculpa.cc, desculpa.cpp ou desculpa.pas

Cuca saiu para jogar futebol com os amigos e esqueceu do encontro que tinha com a namorada. Ciente da mancada, Cuca deseja elaborar um pedido especial de desculpas. Resolveu então enviar flores e usar o cartão da floricultura para escrever um pedido especial de desculpas.

Cuca buscou na internet um conjunto de frases bonitas contendo a palavra 'desculpe' (que pode ocorrer mais de uma vez na mesma frase). No entanto, o cartão da floricultura é pequeno, e nem todas as frases que Cuca colecionou poderão ser aproveitadas.

Cuca quer aproveitar o espaço do cartão, onde cabe um número limitado de caracteres, para escrever um sub-conjunto das frases coletadas de modo que apareça o máximo de vezes possível a palavra 'desculpe'.

Tarefa

Escreva um programa que, dados o número de caracteres que cabem no cartão e a quantidade de frases coletadas (com os respectivos comprimentos e os números de ocorrências da palavra 'desculpe'), determine o número máximo de vezes que a palavra aparece, utilizando apenas as frases colecionadas, sem repetí-las.

Entrada

A entrada é constituída de vários casos de teste. A primeira linha de um caso de teste contém dois números inteiros C e F indicando respectivamente o comprimento do cartão em caracteres ($8 \leq C \leq 1000$) e o número de frases coletadas ($1 \leq F \leq 50$). Cada uma das F linhas seguintes descreve uma frase coletada. A descrição é composta por dois inteiros N e D que indicam respectivamente o número de caracteres na frase ($8 \leq N \leq 200$) e quantas vezes a palavra 'desculpe' ocorre na frase ($1 \leq D \leq 25$). O final da entrada é indicado por $C = F = 0$.

A entrada deve ser lida do dispositivo de entrada padrão (normalmente o teclado).

Saída

Para cada caso de teste seu programa deve produzir três linhas na saída. A primeira identifica o conjunto de teste no formato "`Teste n`", onde `n` é numerado a partir de 1. A segunda linha deve conter o máximo número de vezes que a palavra 'desculpe' pode aparecer no cartão, considerando que apenas frases coletadas podem ser utilizadas, e cada frase não é utilizada mais de uma vez. A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

A saída deve ser escrita no dispositivo de saída padrão (normalmente a tela).

Restrições

$8 \leq C \leq 1000$ ($C = 0$ apenas para indicar o fim da entrada)
$1 \leq F \leq 50$ ($S = 0$ apenas para indicar o fim da entrada)
$8 \leq N \leq 200$
$1 \leq D \leq 25$

Exemplo de Entrada	**Saída para o Exemplo de Entrada**
200 4 100 4 100 1 120 2 80 5 40 3 10 1 10 1 20 2 0 0	Teste 1 9 Teste 2 4

Nível 2 Fase 2

8 Capítulo 8

Revisão
Olimpíada no sábado

9 Capítulo 9

Este capítulo aborda a busca em nível um tema que será base para toda a parte dos grafos e para a resolução dos problemas mais trabalhosos da olimpíada.

9.1 Fila

A fila é uma estrutura de dados usada para ordenar a sequência de chegada e processamento de dados. O nome faz menção a organização de pessoas onde é atendido primeiro quem chegou antes.

A fila é representada como um vetor com um número grandes de posições onde os itens são inseridos na próxima posição vaga do vetor e são lidos seguindo um índice indicando de quem é a vez de ser processado. O código 9.1 mostra a construção de uma fila com as funções de inserir um novo elemento, ler o próximo elemento e verificar se a fila está vazia.

```
1  #include <iostream>
2  using namespace std;
3
4  int fila[1000];
5  int posLeitura = 0;
6  int posEscrita = 0;
7
8  void inserir(int x){
9      posFinal++;
10     fila[posFinal] = x;
11 }
12
13 int proximo(){
14     posLeitura++;
15     return fila[posLeitura - 1];
16 }
17
18 int filaVazia(){
19     if(posLeitura > posEscrita){
20         return 1;
21     }
22     return 0;
23 }
```

Código 9.1: Funções básicas para manipular uma fila.

9.2 Busca em Nível

A busca em nível é um algoritmo que acessa os elementos de um grafo na ordem definida por uma fila. Considere o problema de achar a distância para a saída de uma caverna, da prova de 2005, como demonstra a figura 9.1. A figura 9.2. representa o desenho da figura 9.1 como uma matriz.

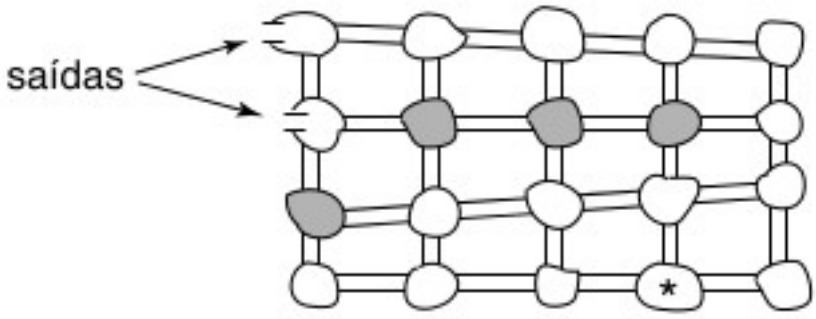


Figura 9.1: Espaços em brancos são lugares onde o duende pode passar. Espaços cinzas são obstáculos onde o doente não pode passar. O asterisco é a posição inicial do duende. Os locais sinalizados como a saída são o destino do duende.

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 2 & 2 & 2 & 1 \\ 2 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 3 & 1 \end{bmatrix}$$

Figura 9.2: Representação matricial do problema do duende perdido. Os 0's representam a saída, 1's representam caminhos abertos, 2 são os obstáculos e 3 é a posição inicial do duende.

Uma busca em nível neste labirinto primeiro preencheria os pontos de distância 1 com o valor 1, depois preencheria os pontos de distância 2 com valor 2, e assim sucessivamente, até chegar ao ponto de saída com o valor da distância procurado. A figura 9.3 mostra essa operação.

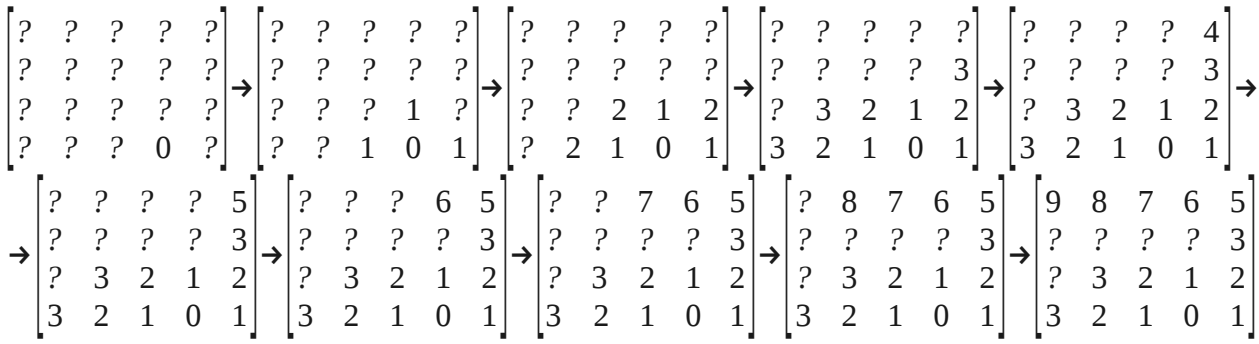


Figura 9.3: A sequência de passos executada por uma busca em nível para encontrar a menor distância para a saída.

O código a seguir realiza as a leitura do valor de entrada e faz a estrutura de dados inicial para a resolução do problema. Programe a busca em nível e encontre a menor distância da saída.

9.3 Problemas

Anterior a 2005

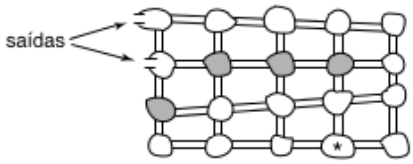
Número de Erdos 2003
Nível 1 Fase 2
Orkut 2004
Nível 2 Fase 1
Desafio cartográfico
http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2011f1p2_cartografico
Nível 2 Fase 2
Móbile http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2007f1p2_mobile

Duende Perdido

Arquivo fonte: duende.c, duende.cc, duende.cpp ou duende.pas

Gugo, o duende, ficou preso em uma caverna e precisa sair o mais rapidamente possível. A caverna é formada por salões interligados por túneis, na forma de uma grade retangular, com N linhas e M colunas. Alguns dos salões da caverna têm paredes de cristal. Duendes, como todos sabem, não gostam de ficar em ambientes com qualquer tipo de cristal, pois seus organismos entram em ressonância com a estrutura de cristais, e em casos extremos os duendes podem até mesmo explodir. Compreensivelmente, Gugo não quer entrar em nenhum salão com parede de cristal.

A figura abaixo mostra uma caverna com quatro linhas e cinco colunas de salões; os salões cinza têm paredes de cristal. A posição inicial de Gugo é indicada com um caractere '*'.



Tarefa

Você deve escrever um programa que, dadas a configuração da caverna e a posição inicial de Gugo dentro da caverna, calcule qual o número mínimo de salões pelos quais o duende deve passar antes de sair da caverna (não contando o salão em que o duende está inicialmente), mas contando o salão que tem saída para o exterior).

Entrada

A caverna será modelada como uma matriz de duas dimensões, cujos elementos representam os salões. Um salão que não tem parede de cristal e que tem saída para o exterior da caverna é representado pelo valor 0; um salão que não tem parede de cristal e não tem saída para o exterior é representado pelo valor 1; um salão que tem parede de cristal é representado pelo valor 2; e o salão em que o duende está inicialmente (que não tem saída para o exterior e nem paredes de cristal) é representado pelo valor 3. A figura abaixo mostra a representação da caverna apresentada acima.

0	1	1	1	1
0	2	2	2	1
2	1	1	1	1
1	1	1	3	1

A primeira linha da entrada contém dois números inteiros N e M que indicam respectivamente o número de linhas ($1 \leq N \leq 10$) e o número de colunas ($1 \leq M \leq 10$) da representação da caverna. Cada uma das N linhas seguintes contém M números inteiros C_i, descrevendo os salões da caverna e a posição inicial do duende ($0 \leq C_i \leq 3$). Você pode supor que sempre há um trajeto que leva Gugo à saída da caverna.

A entrada deve ser lida do dispositivo de entrada padrão (normalmente o teclado).

Saída

Seu programa deve produzir uma única linha na saída, contendo um número inteiro representando a quantidade mínima de salões pelos quais Gugo deve passar antes de conseguir sair da caverna (não contando o salão em que ele está inicialmente, mas contando o salão que tem saída para o exterior).

A saída deve ser escrita no dispositivo de saída padrão (normalmente a tela).

Restrições

$1 \leq N \leq 10$
$1 \leq M \leq 10$
$0 \leq C_i \leq 3$

Exemplos de entrada e saída

Exemplo 1

Entrada	**Saída**
4 5 0 1 1 1 1 0 2 2 2 1 2 1 1 1 1 1 1 1 3 1	8

Exemplo 2

Entrada	**Saída**
1 10 2 0 1 1 3 1 1 1 0 1	3

10 Capitulo 10

Neste capítulo são apresentados dois algoritmos novos. O primeiro e mais simples é o de Prim, seu funcionamento é muito similar ao segundo, que é o de Dijkstra, no entanto é muito mais simples de entender. O algoritmo de Dijkstra busca pelo menor caminho entre vértices de um grafo, como a busca em largura, só que com arestas com distância diferentes. Comparado a busca em nível, os dois algoritmos marcam elementos como visitados até percorrerem todo o grafo, somente é adicionada uma verificação para decidir o próximo vértice mais complexa.

10.1 Representação Matricial de um Grafo

O grafo apresentado no capítulo anterior exemplificava um campo bidimensional com poucas ligações entre os nós. Imagine a necessidade de representar um grafo mais geral como o mostrado na figura a seguir.

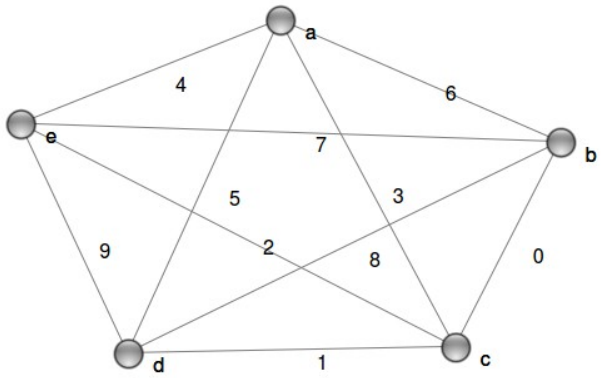


Figura 10.1: Grafo de exemplo para explicar novos conceitos.

Uma matriz representaria esse grafo indicando na posição i e j a distância de i até j.

custo	*a*	*b*	*c*	*d*	*e*
a	0	6	3	5	4
b	6	0	0	8	7
c	3	0	0	1	2
d	5	8	1	0	9
e	4	7	2	9	0

Figura 10.2: Representação matricial do grafo 10.1.

10.2 Prim

O algoritmo de Prim encontra um conjunto de arestas com o menor valor possível que ligam todos os vértices. Conforme já mencionado, inicia-se em alguém vértice qualquer e repete-se os seguintes passos: atualiza os pesos mínimos dos outros vértices que passam pelo último vértice inserido, adiciona o vértice com o menor valor de aresta que ainda não foi visitado. Os passos acabam quando todos os vértices foram percorridos. A figura 10.3 é um exercício para ser feito com lápis e borracha para se familiarizar com o algoritmo de Prim.

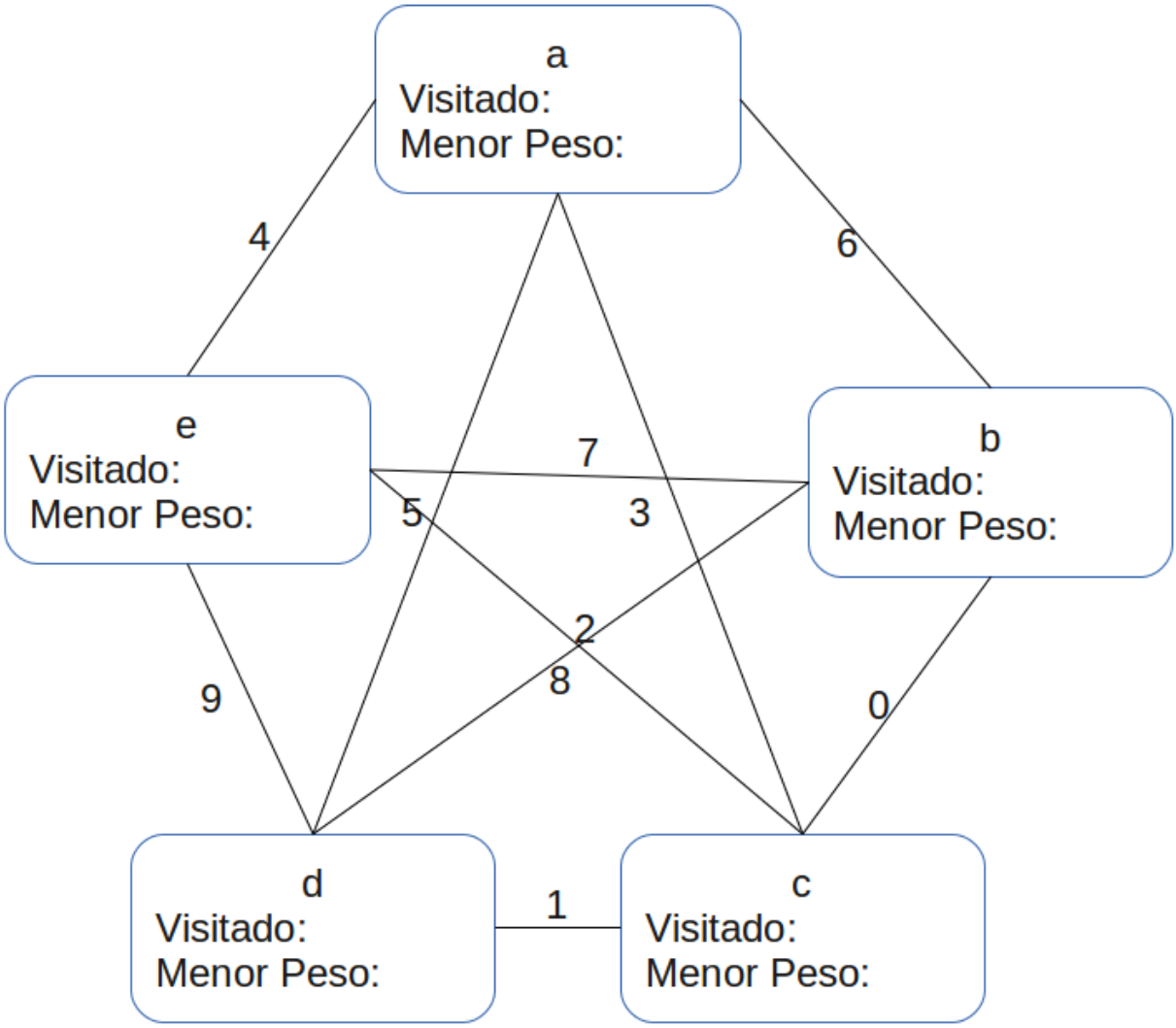


Figura 10.3: Exercício para se familiarizar com o algoritmo de Prim.

10.3 Dijkstra

O algoritmo de Dijkstra encontra o menor caminho entre vértices. A ideia é a cada passo visitar um novo vértice o qual se tenha certeza de conhecer o menor caminho até ele. Dado um conjunto de vértices que já se conhece o menor caminho, o próximo a ser visitado está a apenas uma aresta de distância dos vértices presentes nesse conjunto e possui, obviamente, a menor distância. Para calcular a menor distância atual de cada vértice, atualiza-se a cada passo a distância como o menor valor entre a distância anterior e a distância passando pelo vértice que acabou de ser adicionado. Pratique na figura 10.4 e também na figura 10.5.

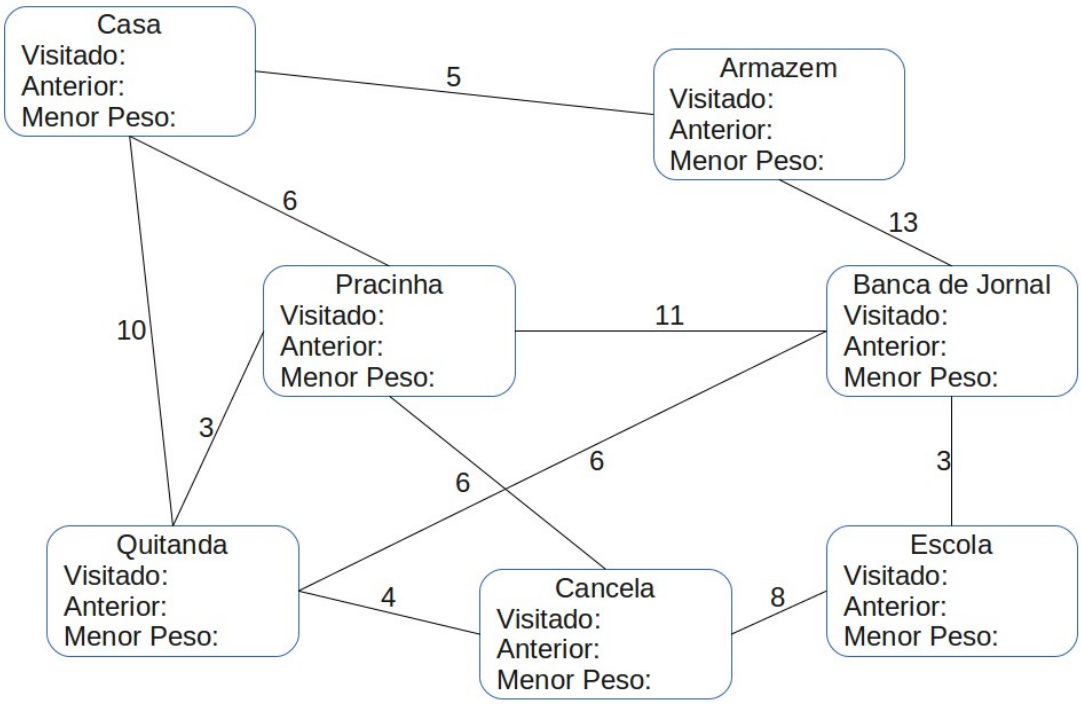


Figura 10.4: Encontre o menor caminho entre a Casa e a Escola.

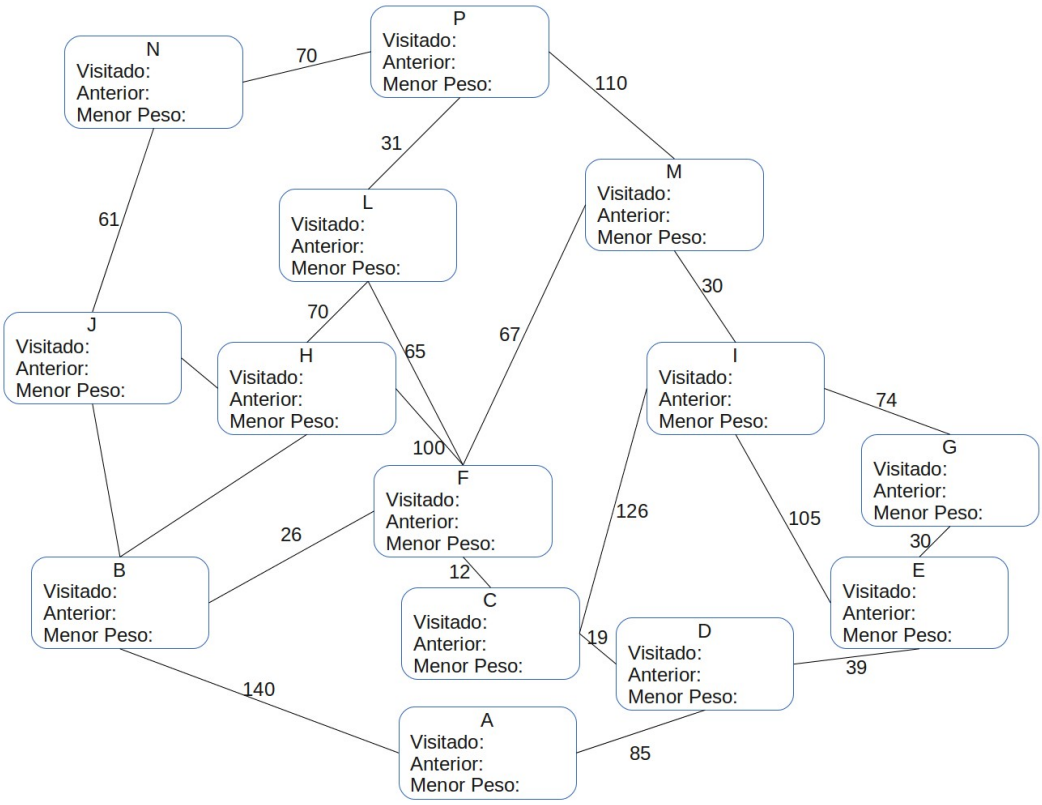


Figura 10.5: Encontre o menor caminho entre o ponto P e o ponto A.

10.4 Problemas

Nível 1 Fase 1
Nível 1 Fase 2
Nível 2 Fase 1
Museu 2006
Nível 2 Fase 2
Caminho das Pontes http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2009f1p2_pontes
Reunião http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2010f1p2_reuniao

11 Busca em Profundidade

A busca em profundidade percorre todos os vértices de um grafo ou alguma outra organização de dados em uma ordem menos útil que a busca em nível. No entanto, uma grande vantagem sobre a busca em nível é a relativa facilidade de programação.

11.1 Função Recursiva

Cada função chamada possui suas próprias variáveis locais. Se uma função chamar a ela mesma várias vezes, então existirão diversas variáveis locais guardando valores diferentes dentro de um programa, observe o código 11.1 e o momento das chamadas das funções recursivas que guardam mais valores no diagrama da figura 11.2.

```
1  #include <iostream>
2
3  using namespace std;
4
5  int fibonacci(int n){
6      if(n == 0 || n == 1)
7          return 1;
8      int soma;
9      soma = fibonacci(n-1);
10     soma += fibonacci(n-2);
11     return soma;
12 }
13
14 int main(){
15     //1, 1, 2, 3, 5
16     cout << fibonacci(4);
17
18
19 }
```

Figura 11.1: Código recursivo para a exibição do resultado do quarto número da sequência de Fibonacci.

```
1  fibonacci(4)
2      int soma = fibonacci(3);
3          int soma = fibonacci(2);
4              int soma = fibonacci(1);
5              soma += fibonacci(0);
6              return soma;
7          soma += fibonacci(1);
8          return soma;
9      soma += fibonacci(2);
10         int soma = fibonacci(1);
11         soma += fibonacci(0);
12         return soma;
13     return soma;
14 return soma;
```

Figura 11.2: Sequência de chamadas da função Fibonacci. Observe que em cada chamada da função existe a variável soma e várias delas residem na memória do computador ao mesmo tempo.

Observe na figura 11.2 que além do programa gravar o valor de variáveis com o mesmo nome nas chamadas recursivas, o ponto de retorno também é guardado em cada chamada de função.

Existem problemas de grafos que é exigido percorrer uma série de vértices contando ou marcando quais vértices foram percorridos sem importar com a ordem. Nestes problemas é mais fácil usar a busca em profundidade do que a em nível.

Nível 1 Fase 1
Colorindo http://olimpiada.ic.unicamp.br/pratique/programacao/nivel1/2011f2p1_colorir
Nível 1 Fase 2
Lobo Mau

Lobo Mau

Nome do arquivo fonte: `lobo.c`, `lobo.cpp`, *ou* `lobo.pas`

Na fazenda do Sr. Amarante existe um certo número de ovelhas. Enquanto elas estão dormindo profundamente, alguns lobos famintos tentam invadir a fazenda e atacar as ovelhas. Ovelhas normais ficariam indefesas diante de tal ameaça, mas felizmente as ovelhas do Sr. Amarante são praticantes de artes marciais e conseguem defender-se adequadamente.

A fazenda possui um formato retangular e consiste de campos arranjados em linhas e colunas. Cada campo pode conter uma ovelha (representada pela letra 'k'), um lobo (letra 'v'), uma cerca (símbolo '#') ou simplesmente estar vazio (símbolo '.'). Consideramos que dois campos pertencem a um mesmo *pasto* se podemos ir de um campo ao outro através de um caminho formado somente com movimentos horizontais ou verticais, sem passar por uma cerca. Na fazenda podem existir campos vazios que não pertencem a nenhum pasto. Um campo vazio não pertence a nenhum pasto se é possível "escapar" da fazenda a partir desse campo (ou seja, caso exista um caminho desse campo até a borda da fazenda).

Durante a noite, as ovelhas conseguem combater os lobos que estão no mesmo pasto, da seguinte forma: se em um determinado pasto houver mais ovelhas do que lobos, as ovelhas sobrevivem e matam todos os lobos naquele pasto. Caso contrário, as ovelhas daquele pasto são comidas pelos lobos, que sobrevivem. Note que caso um pasto possua o mesmo número de lobos e ovelhas, somente os lobos sobreviverão, já que lobos são predadores naturais, ao contrário de ovelhas.

Tarefa

Escreva um programa que, dado um mapa da fazenda do Sr. Amarante indicando a posição das cercas, ovelhas e lobos, determine quantas ovelhas e quantos lobos estarão vivos na manhã seguinte.

Entrada

A entrada contém um único conjunto de testes, que deve ser lido do *dispositivo de entrada padrão* (normalmente o teclado). A primeira linha da entrada contém dois inteiros R e C que indicam o número de linhas ($3 \leq R \leq 250$) e de colunas ($3 \leq C \leq 250$) de campos da fazenda. Cada uma das R linhas seguintes contém C caracteres, representando o conteúdo do campo localizado naquela linha e coluna (espaço vazio, cerca, ovelha ou lobo).

Saída

Seu programa deve imprimir, na *saída padrão*, uma única linha, contendo dois inteiros, sendo que o primeiro representa o número de ovelhas e o segundo representa o número de lobos que ainda estão vivos na manhã seguinte.

Entrada	**Entrada**	**Entrada**
6 6 ...#.. .##v#. #v.#.# #.k#.# .###.# ...### **Saída** 0 2	8 8 .######. #..k...# #.####.# #.#v.#.# #.#.k#k# #k.##..# #.v..v.# .######. **Saída** 3 1	9 12 .###.#####.. #.kk#...#v#. #..k#.#.#.#. #..##k#...#. #.#v#k###.#. #..#v#....#. #...v#v####. .####.#vv.k# ....#### **Saída** 3 5

Nível 2 Fase 1
Transmissão de Energia 2005

Transmissão de Energia

Arquivo fonte: energia.c, energia.cc, energia.cpp ou energia.pas

A distribuição de energia para as diversas regiões do país exige um investimento muito grande em linhas de transmissão e estações transformadoras. Uma linha de transmissão interliga duas estações transformadoras. Uma estação transformadora pode estar interligada a uma ou mais outras estações transformadoras, mas devido ao alto custo não pode haver mais de uma linha de transmissão interligando duas estações.

As estações transformadoras são interconectadas de forma a garantir que a energia possa ser distribuída entre qualquer par de estações. Uma *rota* de energia entre duas estações e_1 e e_k é definida como uma seqüência $(e_1, l_1, e_2, l_2, ...e_{k-1}, l_{k-1}, e_k)$ onde cada e_i é uma estação transformadora e cada l_i é uma linha de transmissão que conecta e_i e_{i+1}.

Os engenheiros de manutenção do sistema de transmissão de energia consideram que o sistema está em estado *normal* se há pelo menos uma rota entre qualquer par de estações, e em estado de *falha* caso contrário.

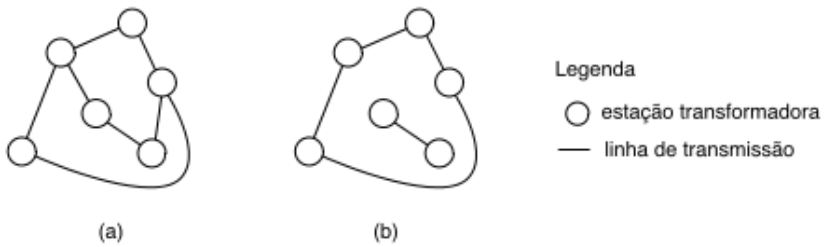


Figura 1: Dois exemplos de sistemas de transmissão: (a) sistema em estado normal; (b) sistema em estado de falha.

Um grande tornado passou pelo país, danificando algumas das linhas de transmissão, e os engenheiros de manutenção do sistema de transmissão de energia necessitam de sua ajuda.

Tarefa

Dada a configuração atual do sistema de transmissão de energia, descrevendo as interconexões existentes entre as estações, escreva um programa que determine o estado do sistema.

Entrada

A entrada é composta de vários casos de teste. A primeira linha de um caso de teste contém dois números inteiros E e L indicando respectivamente o número de estações ($3 \leq E \leq 100$) e o número de linhas de transmissão do sistema ($E - 1 \leq L \leq E \times (E - 1)/2$) que continuam em funcionamento após o tornado. As estações são identificadas por números de 1 a E. Cada uma das L linhas seguintes contém dois inteiros X e Y que indicam que existe uma linha de transmissão interligando a estação X à estação Y. O final da entrada é indicado por $E = L = 0$.

A entrada deve ser lida do dispositivo de entrada padrão (normalmente o teclado).

Saída

Para cada caso de teste seu programa deve produzir três linhas na saída. A primeira identifica o conjunto de teste no formato "`Teste n`", onde `n` é numerado a partir de 1. A segunda linha deve

conter a palavra "`normal`", se, para cada par de estações, houver uma rota que as conecte, e a palavra "`falha`" caso não haja uma rota entre algum par de estações. A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

A saída deve ser escrita no dispositivo de saída padrão (normalmente a tela).

Restrições

$3 \leq E \leq 100$
$E - 1 \leq L \leq E \times (E - 1)/2$

Exemplo de Entrada	**Saída para o Exemplo de Entrada**
6 7 1 2 2 3 3 4 4 5 5 6 6 2 1 5 4 3 1 2 4 2 1 4 0 0	Teste 1 normal Teste 2 falha

Nível 2 Fase 2
Tarzan http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2012f1p2_tarzan

12 Capítulo 12 (Em construção)

Map e programação dinâmica

A programação dinâmica resolve problemas combinando soluções de subproblemas. Estes subproblemas são dependentes entre si ou dependem de um caso base. O termo programação, neste contexto, refere-se ao método tabular e não a escrita de código propriamente. Um algoritmo de programação dinâmica resolve cada subproblema uma única vez e guarda o resultado em uma tabela.

Para exemplificar considere o problema da montagem de um carro no menor tempo possível passando por estações de uma fábrica hipotética.

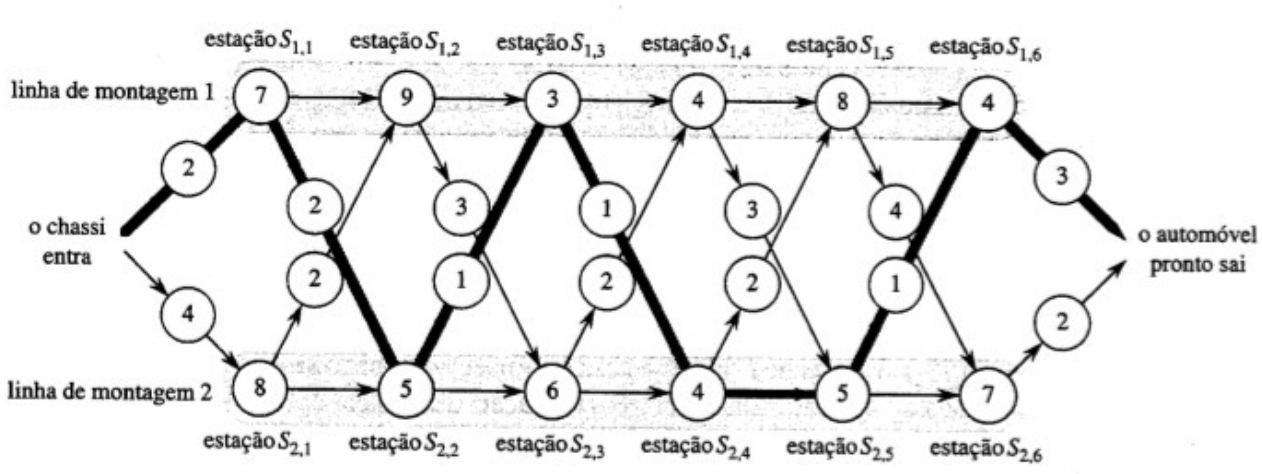


Figura 1: Análise da montagem de um carro no menor tempo possível.
Fonte: Cormem 2002.

A figura 1 apresenta o tempo gasto em cada processo de uma linha de montagem e o tempo para trocar de uma linha de montagem para a outra. O caminho em negrito é o que minimiza o tempo de montagem do carro. Para encontrar esse caminho de forma eficiente basta anotar a melhor solução para cada nó estação da esquerda para a direita. A melhor solução de cada estação é o mínimo entre o peso da estação anterior ou o peso da estação da outra linha de montagem somada com o custo de transferência entre as estações. A tabela a seguir ilustra o peso de cada estação:

	1	2	3	4	5	6
f1[j]	9	18	20	24	32	35
f2[j]	12	16	22	25	30	37

12.1 Problemas

Nível 1 Fase 1

Nível 1 Fase 2

Nível 2 Fase 1
Colheita de Caju 2006
Nível 2 Fase 2

Frequência na aula
http://olimpiada.ic.unicamp.br/pratique/programacao/nivel2/2012f1p2_frequencia

13 Anexos

Para acessar os manuais das funções de programação durante a prova digite man e o nome da função desejada.

13.1 qsort

QSORT(3) Linux Programmer's Manual QSORT(3)

NAME

qsort, qsort_r - sort an array

SYNOPSIS

```
#include <stdlib.h>

void qsort(void *base, size_t nmemb, size_t size,
        int (*compar)(const void *, const void *));

void qsort_r(void *base, size_t nmemb, size_t size,
        int (*compar)(const void *, const void *, void *),
        void *arg);
```

Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

qsort_r(): _GNU_SOURCE

DESCRIPTION

The qsort() function sorts an array with nmemb elements of size size. The base argument points to the start of the array.

The contents of the array are sorted in ascending order according to a comparison function pointed to by compar, which is called with two arguments that point to the objects being compared.

The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted array is undefined.

The qsort_r() function is identical to qsort() except that the comparison function compar takes a third argument. A pointer is passed to the comparison function via arg. In this way, the comparison function does not need to use global variables to pass through arbitrary arguments, and is therefore reentrant and safe to use in threads.

RETURN VALUE

The qsort() and qsort_r() functions return no value.

VERSIONS

qsort_r() was added to glibc in version 2.8.

CONFORMING TO

The qsort() function conforms to SVr4, 4.3BSD, C89, C99.

NOTES

Library routines suitable for use as the compar argument to qsort() include alphasort(3) and versionsort(3). To compare C strings, the comparison function can call strcmp(3), as shown in the example below.

EXAMPLE

For one example of use, see the example under bsearch(3).

Another example is the following program, which sorts the strings given in its command-line arguments:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static int
cmpstringp(const void *p1, const void *p2)
{
   /* The actual arguments to this function are "pointers to
      pointers to char", but strcmp(3) arguments are "pointers
      to char", hence the following cast plus dereference */

   return strcmp(* (char * const *) p1, * (char * const *) p2);
}

int
main(int argc, char *argv[])
{
   int j;

   if (argc < 2) {
    fprintf(stderr, "Usage: %s <string>...\n", argv[0]);
    exit(EXIT_FAILURE);
   }

   qsort(&argv[1], argc - 1, sizeof(argv[1]), cmpstringp);

   for (j = 1; j < argc; j++)
       puts(argv[j]);
   exit(EXIT_SUCCESS);
}
```

SEE ALSO

sort(1), alphasort(3), strcmp(3), versionsort(3)

COLOPHON
This page is part of release 3.40 of the Linux man-pages project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man-pages/.

2012-03-08 QSORT(3)

13.2 sqrt

SQRT(3) Linux Programmer's Manual SQRT(3)

NAME
sqrt, sqrtf, sqrtl - square root function

SYNOPSIS
#include <math.h>

double sqrt(double x);
float sqrtf(float x);
long double sqrtl(long double x);

Link with -lm.

Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

sqrtf(), sqrtl():
_BSD_SOURCE || _SVID_SOURCE || _XOPEN_SOURCE >= 600 ||
_ISOC99_SOURCE || _POSIX_C_SOURCE >= 200112L;
or cc -std=c99

DESCRIPTION
The sqrt() function returns the nonnegative square root of x.

RETURN VALUE
On success, these functions return the square root of x.

If x is a NaN, a NaN is returned.

If x is +0 (-0), +0 (-0) is returned.

If x is positive infinity, positive infinity is returned.

If x is less than -0, a domain error occurs, and a NaN is returned.

ERRORS
See math_error(7) for information on how to determine whether an error

has occurred when calling these functions.

The following errors can occur:

Domain error: x less than -0
errno is set to EDOM. An invalid floating-point exception (FE_INVALID) is raised.

CONFORMING TO
C99, POSIX.1-2001. The variant returning double also conforms to SVr4, 4.3BSD, C89.

SEE ALSO
cbrt(3), csqrt(3), hypot(3)

COLOPHON
This page is part of release 3.40 of the Linux man-pages project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man-pages/.

2010-09-20 SQRT(3)

13.3 pow

POW(3) Linux Programmer's Manual POW(3)

NAME
pow, powf, powl - power functions

SYNOPSIS
#include <math.h>

double pow(double x, double y);
float powf(float x, float y);
long double powl(long double x, long double y);

Link with -lm.

Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

powf(), powl():
_BSD_SOURCE || _SVID_SOURCE || _XOPEN_SOURCE >= 600 ||
_ISOC99_SOURCE || _POSIX_C_SOURCE >= 200112L;
or cc -std=c99

DESCRIPTION
The pow() function returns the value of x raised to the power of y.

RETURN VALUE

On success, these functions return the value of x to the power of y.

If x is a finite value less than 0, and y is a finite noninteger, a domain error occurs, and a NaN is returned.

If the result overflows, a range error occurs, and the functions return HUGE_VAL, HUGE_VALF, or HUGE_VALL, respectively, with the mathematically correct sign.

If result underflows, and is not representable, a range error occurs, and 0.0 is returned.

Except as specified below, if x or y is a NaN, the result is a NaN.

If x is +1, the result is 1.0 (even if y is a NaN).

If y is 0, the result is 1.0 (even if x is a NaN).

If x is +0 (-0), and y is an odd integer greater than 0, the result is +0 (-0).

If x is 0, and y greater than 0 and not an odd integer, the result is +0.

If x is -1, and y is positive infinity or negative infinity, the result is 1.0.

If the absolute value of x is less than 1, and y is negative infinity, the result is positive infinity.

If the absolute value of x is greater than 1, and y is negative infinity, the result is +0.

If the absolute value of x is less than 1, and y is positive infinity, the result is +0.

If the absolute value of x is greater than 1, and y is positive infinity, the result is positive infinity.

If x is negative infinity, and y is an odd integer less than 0, the result is -0.

If x is negative infinity, and y less than 0 and not an odd integer, the result is +0.

If x is negative infinity, and y is an odd integer greater than 0, the result is negative infinity.

If x is negative infinity, and y greater than 0 and not an odd integer, the result is positive infinity.

If x is positive infinity, and y less than 0, the result is +0.

If x is positive infinity, and y greater than 0, the result is positive infinity.

If x is +0 or -0, and y is an odd integer less than 0, a pole error occurs and HUGE_VAL, HUGE_VALF, or HUGE_VALL, is returned, with the same sign as x.

If x is +0 or -0, and y is less than 0 and not an odd integer, a pole error occurs and +HUGE_VAL, +HUGE_VALF, or +HUGE_VALL, is returned.

ERRORS

See math_error(7) for information on how to determine whether an error has occurred when calling these functions.

The following errors can occur:

Domain error: x is negative, and y is a finite noninteger
errno is set to EDOM. An invalid floating-point exception (FE_INVALID) is raised.

Pole error: x is zero, and y is negative
errno is set to ERANGE (but see BUGS). A divide-by-zero floating-point exception (FE_DIVBYZERO) is raised.

Range error: the result overflows
errno is set to ERANGE. An overflow floating-point exception (FE_OVERFLOW) is raised.

Range error: the result underflows
errno is set to ERANGE. An underflow floating-point exception (FE_UNDERFLOW) is raised.

CONFORMING TO

C99, POSIX.1-2001. The variant returning double also conforms to SVr4, 4.3BSD, C89.

BUGS

In glibc 2.9 and earlier, when a pole error occurs, errno is set to EDOM instead of the POSIX-mandated ERANGE. Since version 2.10, glibc does the right thing.

If x is negative, then large negative or positive y values yield a NaN as the function result, with errno set to EDOM, and an invalid (FE_INVALID) floating-point exception. For example, with pow(), one sees this behavior when the absolute value of y is greater than about

9.223373e18.

In version 2.3.2 and earlier, when an overflow or underflow error occurs, glibc's pow() generates a bogus invalid floating-point exception (FE_INVALID) in addition to the overflow or underflow exception.

SEE ALSO

cbrt(3), cpow(3), sqrt(3)

COLOPHON

This page is part of release 3.40 of the Linux man-pages project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man-pages/.

14 Errata e curiosidades

Alguns trechos foram escritos de forma a facilitar o entendimento, mas poderiam ser escritos melhor.

O uso do break deve ser evitado, sempre é possível substitui-lo pelo uso de funções. Sem o uso dos ponteiros necessariamente faria surgir uma variável global e uma chamada de função ou código duplicado. Na olimpíada isso não chega a ser um problema, mas em programas maiores é.

O problema do Duende Perdido do capítulo 9 permite uma solução mais simples. Para preencher a matriz de distâncias ao invés de realizar uma busca em nível bastaria percorrer toda a matriz preenchendo inicialmente todos os salões de adjacentes ao salão de valor 0 com o valor 1, depois percorrer-se-ia toda a matriz novamente preenchendo os salões adjacentes aos de valor 1 com o valor dois e assim sucessivamente. Essa abordagem precisaria percorrer a matriz inúmeras vezes até alcançar a saída e demoraria muito mais que a busca em nível, mas como o tamanho máximo do matriz é de 10x10 então essa é uma solução aceitável.

Não é recomendável o uso de variáveis globais, pois existindo muitas variáveis que podem ser usadas em um determinado local, dificulta a programação. Desta forma em programas grandes deve-se evitar o uso de variáveis globais. As soluções de problemas da OBI gera códigos pequenos, nestas circunstâncias é aceitável o uso de variáveis globais. O uso de vetores ou matrizes como variáveis globais permite modificar seus valores sem precisar passar seu endereço pelo uso de ponteiros, desta forma o uso de ponteiros foi evitado em toda a apostila.