AED III
Estruturas C++
Pair

Pair

Um contêm dois membros: o first e o second. Os elementos do pair podem ser de qualquer tipo. Para declarar, por exemplo, um pair de nome par em que o primeiro elemento é um int e o segundo é uma double, usamos o comando:

pair<int,double> par;

Podemos associar valores aos seus membros, podemos usar o operador "=". Se quiséssemos que o primeiro elemento de par receba o valor "3" e o segundo o valor "2.5", podemos usar os comandos “

par.first=3; par.second=2.5;

Além disso, poderíamos usar a função make_pair(valor1, valor2), que recebe dois parâmetros (valor1 e valor2) e retorna um pair composto por esses dois elementos. Assim, poderíamos fazer par guardar os valores 3 e 2.5 em seus membros com o comando

par=make_pair(3, 2.5);

Uma das grandes utilidades do pair é podermos fazer um outro pair ser um de seus membros. Imagine, por exemplo, que preciso guardar pontos em um plano cartesiano. Para isso, vou usar o primeiro membro de um pair chamado ponto para guardar o número de identificação do ponto. Além disso, quero usar seu segundo membro para guardar suas coordenadas. Para isso, usarei um segundo pair de double, cujo primeiro membro será a coordenada x e o segundo será a coordenada y. Para declararmos tal objeto, usaríamos o comando

pair< int, pair<double, double> > ponto;

Assim, para acessarmos o número de identificação do ponto, usamos "ponto.first". Para acessar o pair que guarda suas coordenadas, usamos "ponto.second", logo, acessamos a coordenada x com "(ponto.second).first" e a coordenada y com "(ponto.second).second". Note que podemos declarar ponto com o primeiro membro "3" e o segundo membro com os números "2.5" e "1.7" com o comando "ponto=make_pair(3, make_pair(2.5, 1.7));".

É importante lembrar que ao usarmos qualquer objeto que use os operadores "<>" na sua declaração, devemos usar espaços para garantir que não escrevamos "<<" ou ">>", visto que esse são operadores próprios da linguagem que executam outras funções. Note, na declaração de ponto, que usou

pair< int, pair<double, double> > ponto;

com espaço entre os ">" do final, ao invés de ""pair<int, pair<double, double>> ponto;", que daria errado pelo uso do operador ">>" no final. Dessa maneira, podemos ter quantos elementos quisermos em um pair, mas a partir de certo número, é muito confuso e cansativo escrevermos "first.second.second.first...", sendo recomendado o uso de uma struct.

Outra grande utilidade do pair é que ele já vem com os operadores de comparação declarados ("<", ">", "<=", ">=" e "=="). Dessa maneira, ele compara dando prioridade ao primeiro membro e usando o segundo apenas como critério de desempate, com os operadores de comparação próprios do tipo.

Imagine, por exemplo, os pares (2, 3) e (1, 7). Temos que (2, 3) > (1, 7), pois o primeiro elemento do primeiro par (2) é maior que o primeiro elemento do segundo par (1). Se os pares fossem (2, 3) e (2, 7), teríamos que (2, 7) > (2, 3), pois o primeiro elemento dos dois pares são iguais e, no critério de desempate, o segundo elemento do segundo par (7) é maior que o segundo elemento do primeiro (3).

Dois pares são iguais ("==" retorna true) somente se todos os elementos de um par forem iguais aos seus correspondentes no outro par. Desse modo, podemos ordenar um vetor de pair usando apenas a função sort sem declararmos uma função de comparação, sabendo que a ordenação irá priorizar o primeiro membro de cada par, usando o segundo como critério de desempate.

Para melhor entendimento, imagine o seguinte problema de ordenação: há n alunos que devem ser ordenados pelo seu desempenho acadêmico. Cada um deles tem seu número de identificação i, a sua nota e o seu número de faltas. Os alunos devem ser ordenados em ordem decrescente de nota.

No caso de empate, deve vir primeiro o aluno com menor número de faltas, e persistindo o empate, o aluno com menor número de identificação deve vir na frente. A primeira linha da entrada contêm um único inteiro o valor de n.

As n linhas seguintes contêm a descrição de cada um dos alunos. Dessa maneira, a i-ésima linha contêm dois inteiros s e f, a nota e o número de faltas do aluno i, respectivamente. A saída deve conter um única linha, com as n identificações dos alunos, ordenadas.

Cada aluno consiste de três informações: nota, faltas e identificação, sendo essa a prioridade no momento de ordenar. Logo, vamos criar um par de um par de inteiros e um inteiro e chamá-lo de aluno, para exemplo.

```
pair< pair< int, int >, int > aluno;
```

Para mantermos a prioridade na ordenação, o primeiro elemento do primeiro par ("aluno.first.first") deve ser a nota, e o segundo elemento do primeiro par ("aluno.first.second") devem ser as faltas. O segundo elemento ("aluno.second") deve ser a identificação do aluno. Entretanto, a função sort ordena de maneira crescente, mas queremos que o aluno de maior nota venha primeiro. Essa situação ocorre muitas vezes na vida de um programador, mas basta salvarmos a nota como um número negativo. Se as notas de dois alunos forem 5 e 3, o sort iria deixá-los na ordem 3, 5. Se salvássemos as notas como números negativos, deixando-as -3, -5, o sort as deixaria na ordem -5, -3, que é a desejada. Basta que na hora de imprimirmos os números, usemos seus opostos, ou seja, imprimiríamos 5 e 3, nessa ordem.

Desse modo, basta criarmos um vector de pair de nome aluno. Essa estrutura será usada na solução do problema. Quando lermos a descrição do aluno i, com sua respectiva nota e número de faltas (s e f), iremos adicionar o par ((-s, f), i) ao vetor. Feito isso basta que o ordenemos com uma chamada do sort, e depois imprimamos, em ordem, os identificadores de seus elementos, que serão seus membros second. Segue o código para melhor entendimento:

```
#include <bits/stdc++.h>
using namespace std;
vector< pair< pair<int, int>, int > > aluno; // declara o vetor que irá guardar os alunos
int n; // declaro o int n
int main()
{
	scanf("%d", &n); // leio o valor de n
	for(int i=1; i<=n; i++) // para cada aluno
	{ // declaro e leio sua nota e seu número de faltas
		int s, f;
		scanf("%d %d", &s, &f);
		// depois o adiciona ao vetor de alunos
		aluno.push_back(make_pair(make_pair(-s, f), i));
	}
	sort(aluno.begin(), aluno.end()); // ordeno os alunos
	// e imprimo suas identificações ordenadas
	for(int i=0; i<n; i++) printf("%d ", aluno[i].second);
	// imprimindo, por fim, a quebra de linha
	printf("\n");
	return 0;
}
```

Exercícios

1. Alguém deixou o quadro de medalhas das olimpíadas fora de ordem. Seu programa deve colocá-lo na ordem correta. A ordem dos países no quadro de medalhas é dada pelo número de medalhas de ouro. Se há empate em medalhas de ouro, a nação que tiver mais medalhas de prata fica a frente. Havendo empate em medalhas de ouro e prata, fica mais bem colocado o país com mais medalhas de bronze. Se dois ou mais países empatarem nos três tipos de medalhas, seu programa deve mostrá-los em ordem alfabética.

Exemplo de Entrada	Exemplo de Saída
8 Belgica 2 2 2 Brasil 7 6 6 Franca 10 18 14 Italia 8 12 8 Australia 8 11 10 Colombia 3 2 3 Suica 3 2 2 Tailandia 2 2 2	Franca 10 18 14 Italia 8 12 8 Australia 8 11 10 Brasil 7 6 6 Colombia 3 2 3 Suica 3 2 2 Belgica 2 2 2 Tailandia 2 2 2

Exercícios

2. O estábulo onde ficam as renas foi intencionalmente aberto pelo Elfo das Trevas, permitindo que cada uma delas corresse e voasse livremente pela fábrica do Papai Noel, causando o maior transtorno. Os elfos estão tentando desesperadamente fazer o possível para deixar o trenó pronto para embarque. Você ficou responsável por colocar cada rena na sua posição correta assim que ela é capturada por um dos outros elfos.

Você sabe que o estábulo segue uma organização baseada na ordem que as renas irão ocupar no trenó. Desta forma, na hora da partida todas podem ser facilmente posicionadas. Diferentemente do que muitos pensam, as renas são posicionadas em uma fila única à frente no trenó. Nem todas as renas do estábulo são utilizadas em cada viagem, isto depende da carga total do trenó.

Exercícios

Você conseguiu a lista com as características que são utilizadas para determinar a ordem de rena. Elas devem ser ordenadas primeiramente de forma decrescente por peso. Caso duas ou mais apresentarem o mesmo peso elas devem ser ordenadas de forma ascendente pela idade, após pela altura e caso ainda persista empate, pelo nome.

Utilizando seu computador mágico de última geração você quer escrever um programa que ordene as renas, de acordo com as características informadas, e exiba somente o número exato de renas que serão utilizadas no trenó (de forma ordenada).

Exemplo de Entrada	Exemplo de Saída
1 9 5 Rudolph 50 100 1.12 Dasher 10 121 1.98 Dancer 10 131 1.14 Prancer 7 142 1.36 Vixen 50 110 1.42 Comet 50 121 1.21 Cupid 50 107 1.45 Donner 30 106 1.23 Blitzen 50 180 1.84	CENARIO {1} 1 - Rudolph 2 - Cupid 3 - Vixen 4 - Comet 5 - Blitzen

Exercícios

3. O professor Odracir organizou junto às suas turmas de Ciência da Computação a confecção de uma camiseta polo que fosse ao mesmo tempo bonita e barata. Após algumas conversas, ficou decidido com os alunos que seriam feitas somente camisetas da cor preta, o que facilitaria a confecção. Os alunos poderiam escolher entre o logo do curso e os detalhes em branco ou vermelho. Assim sendo, Odracir precisa de sua ajuda para organizar as listas de quem quer a camiseta em cada uma das turmas, relacionando estas camisetas pela cor do logo do curso, tamanho (P, M ou G) e por último pelo nome.

Exemplo de Entrada	Exemplo de Saída
3 Maria Joao branco P Marcio Guess vermelho P Maria Jose branco P 0	branco P Maria Joao branco P Maria Jose vermelho P Marcio Guess