#include <iostream>

#include <fstream>

#include <cstdio>

#include <cstdlib>

using namespace std;

bool intercalaBloco(ifstream auxE[2], ofstream auxS[2], int passo, int saida) {

bool intercalou = false;

int valor[2];

int pos[2] = {0,0};

bool valido[2] = {false, false};

while((pos[0] + pos[1]) < (2 \* passo)) {

if((pos[0] < passo) and (not valido[0])) {

if(auxE[0].read((char\*) &valor[0], sizeof(int))) {

valido[0] = true;

} else {

pos[0] = passo;

}

}

if((pos[1] < passo) and (not valido[1])) {

if(auxE[1].read((char\*) &valor[1], sizeof(int))) {

valido[1] = true;

} else {

pos[1] = passo;

}

}

if(valido[0] and valido[1]) {

intercalou = true;

if(valor[0] >= valor[1]) {

auxS[saida].write((const char\*) (&valor[0]), sizeof(int));

valido[0] = false;

pos[0]++;

} else {

auxS[saida].write((const char\*) (&valor[1]), sizeof(int));

valido[1] = false;

pos[1]++;

}

}

else if(valido[0]) {

intercalou = true;

auxS[saida].write((const char\*) (&valor[0]), sizeof(int));

valido[0] = false;

pos[0]++;

}

else if(valido[1]) {

intercalou = true;

auxS[saida].write((const char\*) (&valor[1]), sizeof(int));

valido[1] = false;

pos[1]++;

}

else {

//cout << "circulando, circulando..." << endl;

}

}

return intercalou;

}

void mergeExterno(ifstream &arqEntrada, ofstream &arqSaida) {

ofstream arqB1("arqB1.dat",ios::binary);

ofstream arqB2("arqB2.dat",ios::binary);

if((not arqB1) or (not arqB2)) {

cerr << "Arquivos auxiliares não puderam ser abertos" << endl;

exit(EXIT\_FAILURE);

}

arqEntrada.seekg(0, arqEntrada.end);

int tamanho = arqEntrada.tellg();

tamanho /= 4;

int metade = (tamanho / 2.0) - 0.5;

int valor;

arqEntrada.seekg(0,ios::beg);

for(int i = 0; i < metade; i++) {

arqEntrada.read((char \*) (&valor), sizeof(int));

arqB1.write((char \*) (&valor), sizeof(int));

}

for (int i = metade; i < tamanho; i++) {

arqEntrada.read((char \*) (&valor), sizeof(int));

arqB2.write((char \*) (&valor), sizeof(int));

}

arqB1.close();

arqB2.close();

arqEntrada.close();

ifstream auxEntrada[2];

ofstream auxSaida[2];

int passo = 1;

bool ida = true;

bool ultimo[2];

while(passo <= tamanho) {

if(ida) {

auxEntrada[0].open("arqB1.dat",ios::binary);

auxEntrada[1].open("arqB2.dat",ios::binary);

auxSaida[0].open("arqC1.dat",ios::binary);

auxSaida[1].open("arqC2.dat",ios::binary);

} else {

auxEntrada[0].open("arqC1.dat",ios::binary);

auxEntrada[1].open("arqC2.dat",ios::binary);

auxSaida[0].open("arqB1.dat",ios::binary);

auxSaida[1].open("arqB2.dat",ios::binary);

}

if((not auxEntrada[0]) or (not auxEntrada[1]) or (not auxSaida[0]) or (not auxSaida[1])) {

cerr << "Arquivos auxiliares não puderam ser abertos" << endl;

exit(EXIT\_FAILURE);

}

while((not auxEntrada[0].eof()) and (not auxEntrada[1].eof())) {

ultimo[0] = intercalaBloco(auxEntrada, auxSaida, passo, 0);

ultimo[1] = intercalaBloco(auxEntrada, auxSaida, passo, 1);

}

auxEntrada[0].close();

auxEntrada[1].close();

auxSaida[0].close();

auxSaida[1].close();

ida = not(ida);

passo \*= 2;

}

ifstream auxEnt;

if(ida) {

if(ultimo[0]) {

auxEnt.open("arqB1.dat",ios::binary);

} else {

auxEnt.open("arqB2.dat",ios::binary);

}

} else {

if(ultimo[0]) {

auxEnt.open("arqC1.dat",ios::binary);

} else {

auxEnt.open("arqC2.dat",ios::binary);

}

}

if(not auxEnt) {

cerr << "Arquivos auxiliares não puderam ser abertos" << endl;

exit(EXIT\_FAILURE);

}

while(auxEnt.read((char \*) (&valor), sizeof(int))) {

arqSaida.write((const char\*) (&valor), sizeof(int));

}

auxEnt.close();

remove("arqB1.dat");

remove("arqB2.dat");

remove("arqC1.dat");

remove("arqC2.dat");

}

int main() {

ifstream entrada("Ordenar.dat",ios::binary);

ofstream saida("arquivoOrdenado.dat",ios::binary);

if(not entrada) {

cerr << "Arquivo de entrada não pode ser aberto" << endl;

exit(EXIT\_FAILURE);

}

if(not saida) {

cerr << "Arquivo de saída não pode ser aberto" << endl;

exit(EXIT\_FAILURE);

}

mergeExterno(entrada, saida);

entrada.close();

saida.close();

ifstream Leitura("arquivoOrdenado.dat",ios::binary);

Leitura.seekg(0, Leitura.end);

int tam = Leitura.tellg();

tam /= 4;

int\* valores = new int[tam];

Leitura.seekg(0, Leitura.beg);

Leitura.read((char \*) (valores), tam \* sizeof(int));

Leitura.close();

for (int i = 0; i < tam; i++) {

cout << valores[i] << endl;

}

return 0;

}

#include <iostream>

#include <string>

#include <fstream>

#include <sstream>

struct Dado {

int chave1;

int chave2;

float valor1;

char campo1[8];

char campo2[20];

};

using namespace std;

ostream& operator << (ostream& saida, const Dado& registro) {

saida << registro.chave1 << ' ' << registro.chave2 << ' '

<< registro.valor1 << ' ' << registro.campo1 << ' ' << registro.campo2;

return saida;

}

char MaiusculaAleatoria() {

return char('A' + rand() % 26);

}

float RealAleatorio() {

return float(rand()) / 1.0e10f \* float(rand());

}

void PreencherCampo1(char campo[8]) {

int i = 0;

while (i < 3)

campo[i++] = MaiusculaAleatoria();

while (i < 7)

campo[i++] = char('0' + rand() % 10);

campo[7] = '\0';

}

void PreencherCampo2(char campo[20]) {

static char caracteres[] = " abcdefghijklmnopqrstuvwxyz";

campo[0] = MaiusculaAleatoria();

for (int i = 1; i < 19; ++i)

if (campo[i-1] == ' ')

campo[i] = MaiusculaAleatoria();

else

campo[i] = caracteres[rand() % 28];

campo[19] = '\0';

}

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

bool erroChamada = false;

string modoChamada;

unsigned nroRegistros = 0;

if (argc < 2)

erroChamada = true;

else {

modoChamada = argv[1];

if (modoChamada == "--gerar") {

if (argc < 4)

erroChamada = true;

else {

istringstream iss(argv[2]);

if (not (iss >> nroRegistros))

erroChamada = true;

}

}

else if (modoChamada != "-v")

erroChamada = true;

}

if (erroChamada) {

cerr << "Uso: " << argv[0] << " opÃ§Ãµes\n"

<< "OpÃ§Ãµes disponÃ­veis:\n"

<< " --gerar n nome: gera n registros aleatÃ³rio no arquivo nome.\n"

<< " -v nome: verifica os registros no arquivo nome estÃ£o ordenados.\n"

<< "Exemplo: " << argv[0] << " --gerar 600000 arquivo-original\n";

return 1;

}

srand(unsigned(time(0)));

Dado registro;

if (modoChamada == "--gerar") {

cout << "Gerando " << nroRegistros << " registros no arquivo " << argv[3] << ".\n";

ofstream arquivo(argv[3]);

for (unsigned i = 0; i < nroRegistros; ++i) {

registro.chave1 = rand();

registro.chave2 = rand();

registro.valor1 = RealAleatorio();

PreencherCampo1(registro.campo1);

PreencherCampo2(registro.campo2);

arquivo.write(reinterpret\_cast<char\*>(&registro), sizeof(Dado));

}

}

else {

// verificar se o arquivo estÃ¡ ordenado

Dado r1, r2;

Dado\* ptAnterior = &r1;

Dado\* ptAtual = &r2;

ifstream arquivo(argv[2]);

arquivo.seekg(0, ios::end);

nroRegistros = unsigned(arquivo.tellg() / sizeof(Dado));

cout << "Verificando ordem de " << nroRegistros << " registros.\n";

arquivo.seekg(0);

arquivo.read(reinterpret\_cast<char\*>(ptAnterior), sizeof(Dado));

for (unsigned i = 1; i < nroRegistros; ++i) {

arquivo.read(reinterpret\_cast<char\*>(ptAtual), sizeof(Dado));

// requer a existencia do campo "chave1"

if (ptAnterior->chave1 > ptAtual->chave1) {

cerr << "Erro de ordenaÃ§Ã£o na posiÃ§Ã£o " << i << endl;

return 0;

}

}

cout << "Os registros estÃ£o em ordem.\n";

}

return 0;

}