Referência do Arquivo upload.cpp

```
#include "csv.h"
#include "functions.h"
#include <cmath>
#include <iostream>
```

Definições e Macros

#define	**CSV_IO_NO_THREAD**

Funções

void	**insereNaFolha** (**noArvore** *no, int chave, int P, int quntidadeChaves)
void	**insereNaFolha** (**noArvoreTemp** *tmp, int chave, int P)
void	**insereNoPai** (**noArvore** *no, int chave, int P, int offsetNo)
void	**insere** (int chave, int P)
int	**main** (int argc, char *argv[])

Variáveis

FILE *	**ponteiroArvore**

Definições e macros

◆ CSV_IO_NO_THREAD

#define CSV_IO_NO_THREAD

Funções

◆ insere()

```
void insere ( int  chave,
              int  P
            )
```

Função principal que começa o processo de inserção.

Autores: Erlon, Glenn e Aldemir

```
174 {
175   vetorPais.clear();
176   cabecalhoArvore dadoCabecalho;
177   noArvore noDado;
178   int noAtual;
179   fseek(ponteiroArvore, 0, SEEK_SET);
180   fread(&dadoCabecalho, sizeof(cabecalhoArvore), 1,
181         ponteiroArvore);
182   if (dadoCabecalho.enderecoRaiz == -1)
183   {
184     noArvore noVazio;
185     for (int i = 0; i < QUANTIDADE_PONTEIROS - 1; i++)
186     {
187       noVazio.pares[i].chave = -1;
188       noVazio.pares[i].endereco = -1;
189     }
190     noVazio.ponteiroM = -1;
191     dadoCabecalho.enderecoRaiz = 0;
192     dadoCabecalho.alturaArvore = 1;
193     fwrite(&noVazio, sizeof(noArvore), 1, ponteiroArvore);
194     fseek(ponteiroArvore, 0, SEEK_SET);
195     fwrite(&dadoCabecalho, sizeof(cabecalhoArvore), 1,
196            ponteiroArvore);
197   }
198   else
199   {
200     fseek(ponteiroArvore, 0, SEEK_SET);
201     fread(&dadoCabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
202     fseek(ponteiroArvore, (dadoCabecalho.enderecoRaiz * TAMANHO_BLOCO),
203           SEEK_CUR);
204
205     int nivelAtual = 1;
206     noAtual = dadoCabecalho.enderecoRaiz;
207     while (nivelAtual !=
208            dadoCabecalho.alturaArvore)
209     {
210       int i = 0;
211       fread(&noDado, sizeof(noArvore), 1,
212             ponteiroArvore);
213       vetorPais.push_back(noAtual);
214       while (true)
215       {
216         if (i == QUANTIDADE_PONTEIROS - 1 && noDado.pares[i - 1].chave <
chave)
217         {
218           noAtual = noDado.ponteiroM;
219           fseek(ponteiroArvore, sizeof(cabecalhoArvore) + noAtual * TAMANHO_BL
220                 SEEK_SET);
221           nivelAtual++;
222           break;
223         }
224         else if (noDado.pares[i].chave > chave ||
225                  noDado.pares[i].chave ==
226                      -1)
227         {
228           noAtual = noDado.pares[i].endereco;
229           fseek(ponteiroArvore, sizeof(cabecalhoArvore) + noAtual * TAMANHO_BL
230                 SEEK_SET);
231           nivelAtual++;
232           break;
233         }
234         i++;
235       }
236     }
237   }
238   vetorPais.push_back(noAtual);
```

```
239     fread(&noDado, sizeof(noArvore), 1,
240           ponteiroArvore);
241     int quntidadeChaves = contaChaves(noDado);
242     if (quntidadeChaves < QUANTIDADE_PONTEIROS - 1)
243     {
244       insereNaFolha(&noDado, chave, P, quntidadeChaves);
245       fseek(ponteiroArvore, -sizeof(noArvore), SEEK_CUR);
246       fwrite(&noDado, sizeof(noArvore), 1, ponteiroArvore);
247     }
248     else
249     {
250       noArvore novoNoL;
251       apagaParesNo(&novoNoL);
252       noArvoreTemp tmp;
253       copiaTodosParesNo(&noDado, &tmp);
254       insereNaFolha(&tmp, chave, P);
255 
256       novoNoL.ponteiroM = noDado.ponteiroM;
257       int novoNoOffsetL = dadoCabecalho.quantidadeBlocos + 1;
258       dadoCabecalho.quantidadeBlocos++;
259       noDado.ponteiroM = novoNoOffsetL;
260       apagaParesNo(&noDado);
261       copiaParesNo(&noDado, &tmp, 0, ceil(QUANTIDADE_PONTEIROS / 2.0) - 1); //
    talvez voltar pra inteiro
262       copiaParesNo(&novoNoL, &tmp, ceil(QUANTIDADE_PONTEIROS / 2.0),
    QUANTIDADE_PONTEIROS - 1); // talvez voltar pra inteiro
263       int menoK = novoNoL.pares[0].chave;
264       fseek(ponteiroArvore, -sizeof(noArvore), SEEK_CUR);
265       fwrite(&noDado, sizeof(noArvore), 1, ponteiroArvore);
266       fseek(ponteiroArvore, sizeof(cabecalhoArvore) + novoNoOffsetL *
    TAMANHO_BLOCO, SEEK_SET);
267       fwrite(&novoNoL, sizeof(noArvore), 1, ponteiroArvore);
268       fseek(ponteiroArvore, 0, SEEK_SET);
269       fwrite(&dadoCabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
270 
271       insereNoPai(&noDado, menoK, novoNoOffsetL, noAtual);
272     }
273 }
```

◆ insereNaFolha() [1/2]

```
void insereNaFolha ( noArvore * no,
                     int        chave,
                     int        P,
                     int        quntidadeChaves
                   )
```

Essa função insere na folha sempre quando há espaço no nó

Autor: Aldemir

```
14 {
15   int i;
16   if (chave < (*no).pares[0].chave)
17   {
18     moveParesNo(no, 0, quntidadeChaves);
19     (*no).pares[0].chave = chave;
20     (*no).pares[0].endereco = P;
21   }
22   else
23   {
24     if (quntidadeChaves == 0)
25     {
26       (*no).pares[i + 1].chave = chave;
27       (*no).pares[i + 1].endereco = P;
28     }
29     else
30     {
31       for (i = quntidadeChaves - 1; i >= 0; i--)
32       {
33         if (chave >= (*no).pares[i].chave)
34         {
35           moveParesNo(no, i + 1, quntidadeChaves);
36           (*no).pares[i + 1].chave = chave;
37           (*no).pares[i + 1].endereco = P;
38           break;
39         }
40       }
41     }
42   }
43 }
```

◆ insereNaFolha() [2/2]

```
void insereNaFolha ( noArvoreTemp *  tmp,
                     int             chave,
                     int             P
                   )
```

Função chamada dentro da funcão insere quando a quantidade de chaves não é menor que N-1

Autor: Glenn

```
50 {
51   if (chave < (*tmp).pares[0].chave)
52   {
53     moveParesNo(tmp, 0, QUANTIDADE_PONTEIROS - 1);
54     (*tmp).pares[0].chave = chave;
55     (*tmp).pares[0].endereco = P;
56     return;
57   }
58   for (int i = QUANTIDADE_PONTEIROS - 2; i >= 0; i--)
59   {
60     if (chave >= (*tmp).pares[i].chave)
61     {
62       moveParesNo(tmp, i + 1, QUANTIDADE_PONTEIROS - 1);
63       (*tmp).pares[i + 1].chave = chave;
64       (*tmp).pares[i + 1].endereco = P;
65       break;
66     }
67   }
68 }
```

◆ insereNoPai()

```
void insereNoPai ( noArvore * no,
                   int        chave,
                   int        P,
                   int        offsetNo
                 )
```

Essa funcão é chamada quando não há espaço sufuciente na folha

É feito então um split das folhas, tranferindo a chave a ser inserida para um nó pai.

Se esse processo não for sufuciente, a chave a ser inserida é propagada até chegar na raiz da árvore

Autor: Erlon

```
 79 {
 80   if (vetorPais[0] == offsetNo)
 81   {
 82     noArvore novaRaiz;
 83     apagaParesNo(&novaRaiz);
 84     novaRaiz.pares[0].chave = chave;
 85     novaRaiz.pares[0].endereco = offsetNo;
 86     novaRaiz.pares[1].endereco = P;
 87 
 88     cabecalhoArvore dadoCabecalho;
 89     fseek(ponteiroArvore, 0, SEEK_SET);
 90     fread(&dadoCabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
 91 
 92     dadoCabecalho.quantidadeBlocos += 1;
 93     dadoCabecalho.alturaArvore += 1;
 94     dadoCabecalho.enderecoRaiz = dadoCabecalho.quantidadeBlocos;
 95 
 96     fseek(ponteiroArvore, sizeof(cabecalhoArvore) + TAMANHO_BLOCO * dadoCabeca
 97           SEEK_SET);
 98     fwrite(&novaRaiz, sizeof(noArvore), 1, ponteiroArvore);
 99 
100     fseek(ponteiroArvore, 0, SEEK_SET);
101     fwrite(&dadoCabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
102     return;
103   }
104 
105   int pai = posicaoPai(offsetNo);
106 
107   noArvore dadoPai;
108 
109   fseek(ponteiroArvore, sizeof(cabecalhoArvore) + pai * TAMANHO_BLOCO, SEEK_SE
110   fread(&dadoPai, sizeof(noArvore), 1, ponteiroArvore);
111 
112   int quantidadePonteiro = contaPonteiros(dadoPai);
113 
114   if (quantidadePonteiro < QUANTIDADE_PONTEIROS)
115   {
116     int i = 0;
117     while (dadoPai.pares[i].endereco != offsetNo)
118       i++;
119 
120     if (i == QUANTIDADE_PONTEIROS - 2)
121     {
122       dadoPai.pares[i].chave = chave;
123       dadoPai.ponteiroM = P;
124     }
125     else
126     {
127       dadoPai.pares[i].chave = chave;
128       dadoPai.pares[i + 1].endereco = P;
129     }
130     fseek(ponteiroArvore, -sizeof(noArvore), SEEK_CUR);
131     fwrite(&dadoPai, sizeof(noArvore), 1, ponteiroArvore);
132   }
133   else
134   {
```

```
135     noArvoreTempPai TP;
136     copiaPaiNo(&dadoPai, &TP, chave, P);
137     apagaParesNo(&dadoPai);
138
139     noArvore dadoNovoPai;
140     apagaParesNo(&dadoNovoPai);
141
142     copiaPorPonteiro(&dadoPai, &TP, 0, (ceil(QUANTIDADE_PONTEIROS / 2.0) - 1))
143     int indicePaiK =
144         ceil(QUANTIDADE_PONTEIROS / 2.0) - 1;
145     int paiK = TP.pares[indicePaiK].chave;
146
147     fseek(ponteiroArvore, sizeof(cabecalhoArvore) + TAMANHO_BLOCO * pai, SEEK_
148     fwrite(&dadoPai, sizeof(noArvore), 1, ponteiroArvore);
149
150     copiaPorPonteiro2(&dadoNovoPai, &TP, (ceil(QUANTIDADE_PONTEIROS / 2.0)),
151                       QUANTIDADE_PONTEIROS);
152
153     cabecalhoArvore dadoCabecalho;
154     fseek(ponteiroArvore, 0, SEEK_SET);
155     fread(&dadoCabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
156
157     int dadoPosNovoPai = dadoCabecalho.quantidadeBlocos + 1;
158     fseek(ponteiroArvore, dadoPosNovoPai * TAMANHO_BLOCO, SEEK_CUR);
159     fwrite(&dadoNovoPai, sizeof(noArvore), 1, ponteiroArvore);
160
161     dadoCabecalho.quantidadeBlocos += 1;
162     fseek(ponteiroArvore, 0, SEEK_SET);
163     fwrite(&dadoCabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
164
165     insereNoPai(&dadoPai, paiK, dadoPosNovoPai, pai);
166   }
167 }
```

◆ main()

```
int main ( int      argc,
           char *  argv[]
         )
```

```
276 {
277   bool leitura = true;
278   std::string nomeEntrada = argv[1];
279   ponteiroArvore = fopen("./primarytree.bin", "w+");
280   cabecalhoArvore a_cabecalho;
281   a_cabecalho.alturaArvore = 0;
282   a_cabecalho.quantidadeBlocos = 0;
283   a_cabecalho.enderecoRaiz = -1;
284   fwrite(&a_cabecalho, sizeof(cabecalhoArvore), 1, ponteiroArvore);
285   io::CSVReader<7, io::trim_chars<>, io::double_quote_escape<';', '\"'>> sampl
286   sample.set_header("ID", "Titulo", "Ano", "Autores", "Citacoes", "Atualizacao
287                     "Snippet");
288     FILE *ponteiroHash = fopen("./hash.bin", "r+");
289 
290     if(ponteiroHash == NULL){
291         std::cout << "Error create hash.bin\n";
292         return 0;
293     }
294 
295     for(int i = 0; i < QUANTIDADE_BUCKETS ; i++){
296         tipoBloco bloco;
297         fwrite(&bloco, TAMANHO_BLOCO, 1, ponteiroHash);
298     }
299 
300   tipoArtigoLeitura *ta_aux =
301       (tipoArtigoLeitura *)malloc(sizeof(tipoArtigoLeitura));
302   while (leitura)
303   {
304     try
305     {
306       if (leitura = sample.read_row(ta_aux->ID, ta_aux->Titulo, ta_aux->Ano,
307                                     ta_aux->Autores, ta_aux->Citacoes,
308                                     ta_aux->Atualizacao, ta_aux->Snippet))
309       {
310         int id = std::stoi(ta_aux->ID);
311         insere(id, funcaoHash(id));
312 
313         tipoArtigo artigo;
314         tipoBloco bloco;
315 
316         artigo.ID = stoi(ta_aux->ID);
317         strcpy(artigo.Titulo, ta_aux->Titulo.c_str());
318         artigo.Ano = stoi(ta_aux->Ano);
319         strcpy(artigo.Autores, ta_aux->Autores.c_str());
320         artigo.Citacoes = stoi(ta_aux->Citacoes);
321         strcpy(artigo.Atualizacao, ta_aux->Atualizacao.c_str());
322         strcpy(artigo.Snippet, ta_aux->Snippet.c_str());
323 
324 
325         fseek(ponteiroHash, funcaoHash(artigo.ID)*TAMANHO_BLOCO, SEEK_SET);
326         fread(&bloco, TAMANHO_BLOCO, 1, ponteiroHash);
327         bloco.vetorArtigos[bloco.quantidadeArtigos] = artigo;
328         bloco.quantidadeArtigos++;
329         fseek(ponteiroHash, funcaoHash(artigo.ID)*TAMANHO_BLOCO, SEEK_SET);
330         fwrite(&bloco, TAMANHO_BLOCO, 1, ponteiroHash);
331 
332         std::cout << "Inserindo id: " << ta_aux->ID << std::endl;
333       }
334     }
335     catch (io::error::too_few_columns) {}
336     catch (io::error::escaped_string_not_closed) {}
337   }
338   free(ta_aux);
339   fclose(ponteiroArvore);
340   fclose(ponteiroHash);
341 
342   return 0;
343 }
```

Variáveis

◆ ponteiroArvore

FILE* ponteiroArvore