UNIVERSIDADE FEDERAL DE SÃO CARLOS

Organização e Recuperação da Informação

Trabalho I

Documentação

Cassiano Maia RA: 726507

Julia Milani RA: 726552

Prof. Dr. Ricardo Rodrigues Ciferri

SÃO CARLOS, 07 DE NOVEMBRO DE 2017

1. DESCRIÇÃO

O programa desenvolvido tem como objetivo gerenciador arquivos, onde é possível a inserção, listagem, busca, remoção e compactação de registros, estes divididos em blocos de 512 bytes cada. A linguagem de desenvolvimento é a linguagem C e a função do gerenciador implementado é o catálogo de violinos para venda.

2. IMPLEMENTAÇÃO

Quanto ao tamanho das estruturas para armazenagem de dados, elas foram implementadas com blocos de tamanho fixo de 512 bytes, registros de 64 bytes com os campos: “código” (chave), “ano” e “valor” de 4 bytes e “descrição” de 50 bytes (como mostra a imagem 2.1) e um header de 4 bytes. Dessa maneira, podemos colocar até 8 registros dentro de um bloco, minimizando a sobra de espaço em cada um deles.

```
7   // Definição dos tamanhos fixos do bloco, registro e blocos
8   #define tamCod 4
9   #define tamDesc 50
10  #define tamAno 4
11  #define tamValor 4
12  #define tamReg 64
13  #define tamBloco 512
14  #define tamHeader 4
15
16
17  // Struct que define registros de 62 bytes para um arquivo de catalogo de violinos // 64bytes size
18  typedef struct {
19          int code;
20          char desc[50];
21          int ano;
22          float valor;            // 4 + 4 + 50 + 4 = 62
23  } reg;
```

Figura 2.1: Estruturas para armazenar dados

Foram implementadas também 10 funções necessárias para o funcionamento do gerenciador, sendo elas divididas entre funções de arquivo, de bloco e de registro. As funções são mostradas à seguir:

1. criaBloco(): Cria um novo bloco de 512 bytes. [**COLOCAR PRINT DOS MÉTODOS ATUALIZADOS CASO VOCÊ MUDE ELES INTERNAMENTE DEPOIS]**

```
bloco* criaBloco(){
        bloco* novo = (bloco*)malloc(sizeof(bloco));
        memset(novo,0,tamBloco);
        strncpy(novo->header, "#BLK", 4);
        return novo;
}

```

2. criaBlocoInicial(): Cria o primeiro dos blocos para armazernar registros.

```
blocoinicial* criaBlocoInicial(){
        blocoinicial* novo = (blocoinicial*)malloc(sizeof(blocoinicial));
        memset(novo,0,tamBloco);
        strncpy(novo->header, "#BLK", 4);
        novo->nblocos = 1;
        novo->nregistros = 0;
        return novo;
}

```

3. criaArquivo(): Cria um novo arquivo para ser utilizado.

```
int criaArquivo(){
        FILE* arquivo = fopen("arquivo.txt", "wb");
        if(!arquivo){
                return 0;
        }else{
                blocoinicial* bloco= criaBlocoInicial();
                fwrite(bloco, tamBloco, 1, arquivo);
                free(bloco);
                fclose(arquivo);
                return 1;
        }
}
```

4. AtualizaHeader(FILE*, int, int): Atualiza o header do arquivo e recebe como

parâmetro, respectivamente, o endereço onde está o arquivo, o número de registros e o número de blocos.

```
45  void AtualizaHeader(FILE* arquivo, int nregistros, int nblocos){
46          blocoinicial* primeirobloco = (blocoinicial*)malloc(sizeof(blocoinicial));
47          fseek(arquivo, 0, SEEK_SET);
48          fread(primeirobloco, tamBloco, 1, arquivo);
49          primeirobloco->nblocos += nblocos;
50          primeirobloco->nregistros += nregistros;
51          fseek(arquivo, 0, SEEK_SET);
52          fwrite(primeirobloco, tamBloco, 1, arquivo);
53          free(primeirobloco);
54  }
```

5. compactaArquivo(): Operação de compactação do arquivo, remove os espaços inutilizados dentro dele para a minimização de desperdício dentro do mesmo.

```
void compactaArquivo(){
    FILE* arquivo = fopen("arquivo.txt", "rb+");
    criaTempArquivo();
    FILE* temparquivo = fopen("temparquivo.txt", "rb+");
    bloco* temp = criaBloco();
    blocoinicial* tempinicial = criaBlocoInicial();
    int blocon = 0;
    int regn = 0;

    if(!arquivo){
        printf("Arquivo nao encontrado!\n");
    }else{
        printf("Arquivo encontrado\n");
        //Leitura do bloco inicial
        fread(tempinicial, tamBloco,1,arquivo);
        if((tempinicial->header[0]=='#')&&(tempinicial->header[1]=='B')&&(tempinicial->header[2]=='L')&&(tempinicial->header[3]=='K')){
            while(regn <= 6){
                if(tempinicial->index[regn].code <= 0 ){
                    regn++;
                }else{
                    //Passagem dos registros validos do bloco original para o novo arquivo
                    insereReg(tempinicial->index[regn], temparquivo);
                    regn++;
                }
            }
            blocon++;
            regn = 0;
        }
        //Leitura dos demais blocos
        while ((fread(temp,tamBloco,1,arquivo)) != 0){
            if((temp->header[0]=='#')&&(temp->header[1]=='B')&&(temp->header[2]=='L')&&(temp->header[3]=='K')){
                while(regn <= 6){
                    if(temp->index[regn].code <= 0 ){
                        regn++;
                    }else{
                        insereReg(temp->index[regn], temparquivo);
                        regn++;
                    }
                }
                blocon++;
                regn = 0;
            }else{
                printf("Inconsistencia de dados detectada, o arquivo foi corrompido.\n");
            }
        }
        }
        //Substituição do arquivo original pelo novo arquivo gerado sem fragmentações
        remove("arquivo.txt");
        rename("temparquivo.txt","arquivo.txt");
        free(tempinicial);
        free(temp);
        fclose(arquivo);
        fclose(temparquivo);
    }
}
```

6. insereReg(reg, FILE*): Inserção de registros, recebe como parâmetro um

novo registrador e o endereço do arquivo.

```
int insereReg(reg newreg, FILE* arquivo){
        fseek(arquivo, 0, SEEK_SET);
        int blocon = 0; //para avançar entre os blocos
        int regn = 0;
        bloco* temp = criaBloco(); //bloco temporario para escrevermos o registro
        blocoinicial* tempinicial = criaBlocoInicial();

        if(!arquivo){
                printf("Arquivo nao encontrado!\n");
                return 0;
        }else{
                printf("Arquivo encontrado\n");
                fread(tempinicial, tamBloco,1,arquivo);
                if((tempinicial->header[0]=='#')&&(tempinicial->header[1]=='B')&&(tempinicial->header[2]=='L')&&(tempinicial->header[3]=='K')
                        printf("Bloco validado.\n");
                                while(regn <= 6){
                                        printf("Procurando registro vazio.\n");
                                        if(tempinicial->index[regn].code == 0 || tempinicial->index[regn].code == -1){ // zero para vazio | -
                                                printf("Escrevendo dados.\n");
                                                tempinicial->index[regn] = newreg;
                                                fseek(arquivo, blocon*tamBloco, SEEK_SET);
                                fwrite(tempinicial, tamBloco, 1, arquivo);
                                AtualizaHeader(arquivo, 1, 0);
                                free(temp);
                                free(tempinicial);
                                                return 1;
                                        }else{
                                                printf("Procurando prox registro.\n");
                                                regn++;
                                        }
                                }
                                blocon++;
                                regn = 0;
                }
                while ((fread(temp,tamBloco,1,arquivo)) != 0){
                        if((temp->header[0]=='#')&&(temp->header[1]=='B')&&(temp->header[2]=='L')&&(temp->header[3]=='K')){

                                printf("Bloco validado.\n");
                                while(regn <= 6){
                                        printf("Procurando registro vazio.\n");
                                        if(temp->index[regn].code == 0 || temp->index[regn].code == -1){ // zero para vazio | -1 para registr
                                                printf("Escrevendo dados.\n");
                                                temp->index[regn] = newreg;
                                                fseek(arquivo, blocon*tamBloco, SEEK_SET);
                                fwrite(temp, tamBloco, 1, arquivo);
                                AtualizaHeader(arquivo, 1, 0);
                                free(temp);
                                free(tempinicial);
                                                return 1;
                                        }else{
                                                printf("Procurando prox registro.\n");
                                                regn++;
                                        }
                                }
                                blocon++;
                                regn = 0;
                         }else{
                                printf("Inconsistencia de dados detectada, o arquivo foi corrompido.\n");
                                return 0;
                        }
                }
                printf("Todos os blocos estão cheios.\n");
                temp = criaBloco();
                temp->index[0] = newreg;
                fseek(arquivo, blocon*tamBloco, SEEK_SET);
        fwrite(temp, tamBloco, 1, arquivo);
        AtualizaHeader(arquivo, 1, 1);
        free(temp);
        free(tempinicial);
                return 1;
        }
}
```

7. escreveReg(reg): Escreve em um registro, então, naturalmente, o recebe por parâmetro.

```
153  void escreveReg(reg regout){            //escreve o conteudo de um registro
154          printf("\nConteudo do registro:\n"
155                     "Codigo: %d\n"
156                     "Descrição: %s\n"
157                     "Ano: %d\n"
158                     "Valor: %f\n\n", regout.code, regout.desc, regout.ano, regout.valor);
159  }
160
```

8. removeReg(int): Remove um registro do arquivo, recebendo como parâmetro o index dos registros.

```
162  int removeReg(int rindex){
163          int blocon = 0; //para avançar entre os blocos
164          int regn = 0;
165          FILE* arquivo = fopen("arquivo.txt", "rb+");
166          bloco* temp = criaBloco(); //bloco temporario para escrevermos o registro
167          blocoinicial* tempinicial = criaBlocoInicial();
168          if(!arquivo){
169                  printf("Arquivo nao encontrado!\n");
170                  return 0;
171          }else{
172                  printf("Arquivo encontrado\n");
173                  fread(tempinicial, tamBloco,1,arquivo);
174                  if((tempinicial->header[0]=='#')&&(tempinicial->header[1]=='B')&&(tempinicial->header[2]=='L')&&(tempinicial->header[3]=='K')
175                          printf("Bloco validado.\n");
176                          while(regn <= 6){
177                                  printf("Procurando o registro a ser removido.\n");
178                                  if(tempinicial->index[regn].code == rindex){
179                                          printf("Removendo o registro.\n");
180                                          tempinicial->index[regn].code = -1;
181                                          fseek(arquivo, blocon*tamBloco, SEEK_SET);
182                          fwrite(tempinicial, tamBloco, 1, arquivo);
183                          AtualizaHeader(arquivo, -1, 0);
184                          free(tempinicial);
185                          free(temp);
186                          fclose(arquivo);
187                                          return 1;
188                                  }else{
189                                          printf("Procurando prox registro.\n");
190                                          regn++;
191                                  }
192                          }
193                          blocon++;
194                          regn = 0;
195                  }
196                  while ((fread(temp,tamBloco,1,arquivo)) != 0){
197                          if((temp->header[0]=='#')&&(temp->header[1]=='B')&&(temp->header[2]=='L')&&(temp->header[3]=='K')){
```

```
                        if((temp->header[0]=='#')&&(temp->header[1]=='B')&&(temp->header[2]=='L')&&(temp->header[3]=='K')){
                                printf("Bloco validado.\n");
                                while(regn <= 6){
                                        printf("Procurando o registro a ser removido.\n");
                                        if(temp->index[regn].code == rindex){
                                                printf("Removendo o registro.\n");
                                                temp->index[regn].code = -1;
                                                fseek(arquivo, blocon*tamBloco, SEEK_SET);
                                fwrite(temp, tamBloco, 1, arquivo);
                                AtualizaHeader(arquivo, -1, 0);
                                free(tempinicial);
                                free(temp);
                                fclose(arquivo);
                                                return 1;
                                        }else{
                                                regn++;
                                        }
                                }
                                blocon++;
                                regn = 0;
                        }else{
                                printf("Inconsistencia de dados detectada, o arquivo foi corrompido.\n");
                                return 0;
                        }
                }
                printf("Registro nao encontrado.\n");
                free(tempinicial);
                free(temp);
        fclose(arquivo);
                return 0;
        }
}

```

9. procuraReg(): Procura um registro específico dentro do bloco.

```
int procuraReg(int key){
        FILE* arquivo = fopen("arquivo.txt", "rb+");
        bloco* temp = criaBloco();
        blocoinicial* tempinicial = criaBlocoInicial();
        int blocon = 0; //para avançar entre os blocos
        int regn = 0;

        if(!arquivo){
                printf("Arquivo nao encontrado!\n");
                return 0;
        }else{
                printf("Arquivo encontrado\n");
                fread(tempinicial, tamBloco,1,arquivo);
                if((tempinicial->header[0]=='#')&&(tempinicial->header[1]=='B')&&(tempinicial->header[2]=='L')&&(tempinicial->header[3]=='K')
                        printf("Bloco validado.\n");
                        while(regn <= 6){
                                printf("Procurando o registro.\n");
                                if(tempinicial->index[regn].code == key){
                                        escreveReg(tempinicial->index[regn]);
                        fclose(arquivo);
                        free(tempinicial);
                        free(temp);
                        fclose(arquivo);
                                        return 1;
                                }else{
                                        regn++;
                                }
                        }
                        blocon++;
                        regn = 0;
                }
                while ((fread(temp,tamBloco,1,arquivo)) != 0){
                        if((temp->header[0]=='#')&&(temp->header[1]=='B')&&(temp->header[2]=='L')&&(temp->header[3]=='K')){
                                printf("Bloco validado.\n");
                                while(regn <= 6){
                                        printf("Procurando o registro.\n");
```

```
266                                     if(temp->index[regn].code == key){
267                                             escreveReg(temp->index[regn]);
268                                 fclose(arquivo);
269                                 free(temp);
270                                 fclose(arquivo);
271                                             return 1;
272                                     }else{
273                                             regn++;
274                                     }
275                                 }
276                                 blocon++;
277                                 regn = 0;
278                         }else{
279                                 printf("Inconsistencia de dados detectada, o arquivo foi corrompido.\n");
280                                 return 0;
281                         }
282                 }
283                 printf("Registro nao encontrado.\n");
284                 free(temp);
285         fclose(arquivo);
286         return 0;
287         }
288 }
289
```

10. listaReg(): Lista os registros dentro de um determinado bloco.

```
290 int listaReg(){
291         FILE* arquivo = fopen("arquivo.txt", "rb+");
292         bloco* temp = criaBloco();
293         blocoinicial* tempinicial = criaBlocoInicial();
294         int blocon = 0; //para avançar entre os blocos
295         int regn = 0;
296
297         if(!arquivo){
298                 printf("Arquivo nao encontrado!\n");
299                 return 0;
300         }else{
301                 printf("Arquivo encontrado\n");
302                 fread(tempinicial, tamBloco,1,arquivo);
303                 if((tempinicial->header[0]=='#')&&(tempinicial->header[1]=='B')&&(tempinicial->header[2]=='L')&&(tempinicial->header[3]=='K')
304                         printf("Bloco validado.\n");
305                         while(regn <= 6){
306                                 printf("Procurando o registro.\n");
307                                 if(tempinicial->index[regn].code <= 0 ){
308                                         regn++;
309                                 }else{
310                                         escreveReg(tempinicial->index[regn]);
311                                         regn++;
312                                 }
313                         }
314                         blocon++;
315                         regn = 0;
316                 }
317                 while ((fread(temp,tamBloco,1,arquivo)) != 0){
318                         if((temp->header[0]=='#')&&(temp->header[1]=='B')&&(temp->header[2]=='L')&&(temp->header[3]=='K')){
319                                 printf("Bloco validado.\n");
320                                 while(regn <= 6){
321                                         printf("Procurando o registro.\n");
322                                         if(temp->index[regn].code <= 0 ){
323                                                 regn++;
324                                         }else{
325                                                 escreveReg(temp->index[regn]);
```

```
                                        regn++;
                                    }
                                }
                                blocon++;
                                regn = 0;
                        }else{
                                printf("Inconsistencia de dados detectada, o arquivo foi corrompido.\n");
                                return 0;
                        }
                }
                free(tempinicial);
                free(temp);
        fclose(arquivo);
        return 1;
        }
}
```

11. registroaleatorio(): Gera um registro aleatório.

```
//Função que gera um registro aleatório para inserção em lotes
reg registroaleatorio(){
    reg regin;
    regin.code = (rand()%100) + 1;
    regin.ano = rand()%410 + 1600; // Recebe um ano acima de 1600, ano em que o primeiro violino foi dat
    regin.valor = rand()%10000 + 320; //Recebe um valor acima de 320, que é o valor de um violino estuda
    switch (rand()%10 + 1){
        case 1:
            strcpy(regin.desc, "Antonius Stradivarius Cremonenfis, Faciebat");
        break;
        case 2:
            strcpy(regin.desc, "Jean Baptiste Vuillaume a Paris, Rue Croix des Pet");
        break;
        case 3:
            strcpy(regin.desc, "Copy of Antonius Stradivarius, made in Czech Rep.");
        break;
        case 4:
            strcpy(regin.desc, "Francesco Ruggieri detto il per Cremona");
        break;
        case 5:
            strcpy(regin.desc, "G. Carlettinius fec. Centum");
        break;
        case 6:
            strcpy(regin.desc, "Enrico Orselli, Liutaio, Pesaro");
        break;
        case 7:
            strcpy(regin.desc, "T. J. Holder, Luthier, Paris, Model 4/4");
        break;
        case 8:
            strcpy(regin.desc, "Johann Glass, Getgenmacher in Lelpzig");
        break;
        case 9:
            strcpy(regin.desc, "Giovanni Leoni Filius, Parmo n 1432, 4/4");
        break;
        case 10:
            strcpy(regin.desc, "Luigi Gambelimberti, fece a Seveso l'anno 1920");
        break;
        case 11:
            strcpy(regin.desc, "Georges Defat, Luthier a Paris, anno 1900");
        break;
    }
    return regin;
}
```

O código contém os comentários detalhados sobre todos os métodos.