Relatório em resposta à correção do 1o trabalho prático
Maria Luisa Santos Rodrigues

Resumo das alterações:

1) Alterei a chamada da função clearAllData() para que ela só seja chamada caso o arquivo de hash esteja vazio;
2) Alterei a função printFile para exibir os apontadores para as páginas como se fossem contadas a partir de 1 ao invés de a partir de 0;
3) Alterei a função printFile para imprimir null ao final de todas as páginas; antes, caso a página estivesse só parcialmente preenchida, o programa imprimia os registros vazios. Agora não os imprime mais, só imprime null.

Incluí no .zip o arquivo original hashing.cpp caso haja necessidade. O programa alterado é o arquivo hashingnew.cpp.
Para limpar os registros salvos basta apagar o conteúdo do arquivo hash.bin, porém não apague o arquivo em si!

Descrição dos problemas, alterações e printscreens das saídas do programa:

Todos os testes do programa detalhados nesse relatório foram feitos no Ubuntu com os seguintes comandos, sendo testeXXa e testeXXb os pares de arquivos de entrada fornecidos pelo professor:

```
g++ hashingnew.cpp
./a.out < testeXXa
./a.out < testeXXb
```

1) Primeiramente, houve um erro da minha parte na linha 363 do trabalho, que era uma chamada à função clearAllData(). Essa função foi escrita como uma função de debug que limpava os arquivos que o programa usava e acabei esquecendo de comentar a chamada à ela no arquivo cpp final. Sem comentá-la, cada execução do programa estava encontrando arquivos sem nenhum registro e por isso as saídas eram incorretas.
O programa espera um hash.bin e file.bin que não estejam vazios, mesmo que não haja nenhum registro salvo, então implementei uma pequena função que checa se eles estão vazios e os reinicia com um heap vazio caso estejam. Assim não precisa ficar comentando e descomentando clearAllData().

A função é a seguinte:

```
354  int getHashFileSize(){
355      int n = 0;
356
357      FILE* hashFile;
358      hashFile = fopen("hash.bin", "r+");
359
360      if (hashFile == null) {
361          fclose(hashFile);
362          fopen("hash.bin", "w+"); fclose(hashFile);
363          return 0; }
364      while (fgetc(hashFile) != EOF) ++n;
365
366      fclose(hashFile);
367      return n;
368  }
369
370  int main(){
371      // iniciar variáveis globais:
372      REG_SIZE = sizeof(reg_t);
373      //cout << "\n>> BEGINNING OF PROGRAM <<\n";
374      if (getHashFileSize() < 40){
375              // 40 bytes é o tamanho do hash.bin quando o heap está vazio,
376              // então é o menor tamanho possível
377          //cout << "Hash file is empty, clearing all data...\n";
378          clearAllData();
379      }
380
```

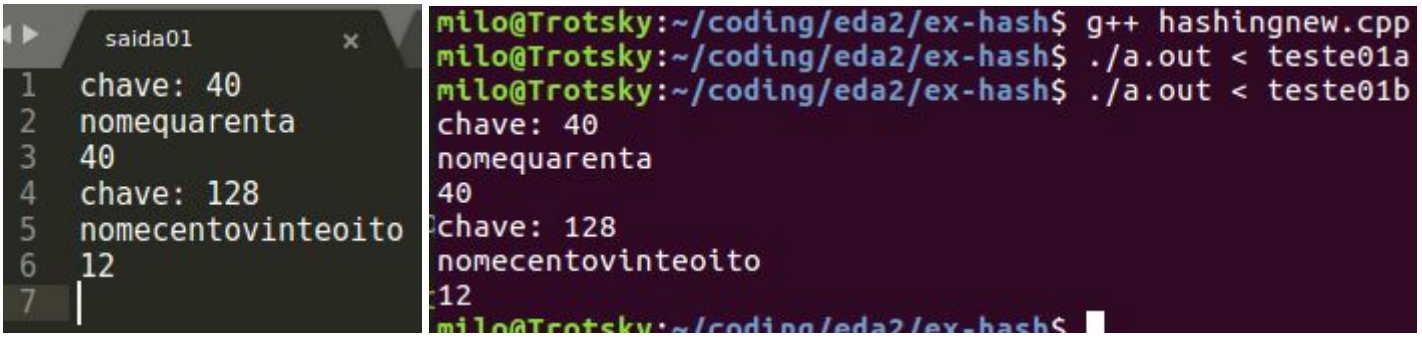
2) As páginas no meu programa são numeradas a partir de 0, porém aparentemente a correção esperava páginas numeradas a partir de 1. Implementei as páginas a partir de 0 pois elas ficam armazenadas num vector (quando estão na memória principal) e a indexação em 0 facilita o acesso. É possível fazer com que o comando p (imprimir arquivo) imprima o índice das páginas indexando em 1, apenas somando 1 ao índice que a função printFile originalmente encontra. Fiz essa alteração na função printFile; é uma alteração meramente estética e não altera o funcionamento do programa.

3) Ainda sobre o comando p, meu programa só imprime null quando encontra uma página totalmente vazia, que foi o que entendi pela especificação do trabalho. Porém as saídas fornecidas pelo professor imprimem null quando encontram um registro vazio (mesmo que a página não esteja totalmente vazia) e/ou ao final da página caso ela esteja cheia. Isso pode ser facilmente modificado: basta que, na função de imprimir arquivo, ao se deparar com um registro vazio (chave = 0, nome = “” e idade = 0) ou chegar ao fim da página, o programa imprima null e pare de varrer a página.

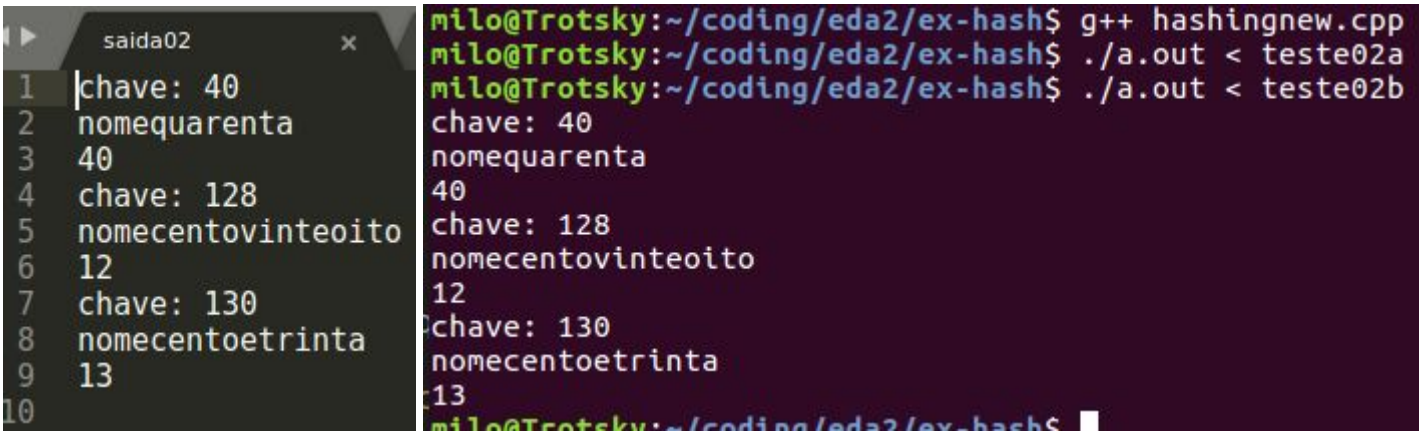
Comparação das saídas do programa com as saídas fornecidas pelo professor:

Os arquivos saidaXX são as saídas fornecidas pelo professor, abertas no Sublime Text. A outra parte de cada imagem é um printscreen do terminal após a execução do programa.

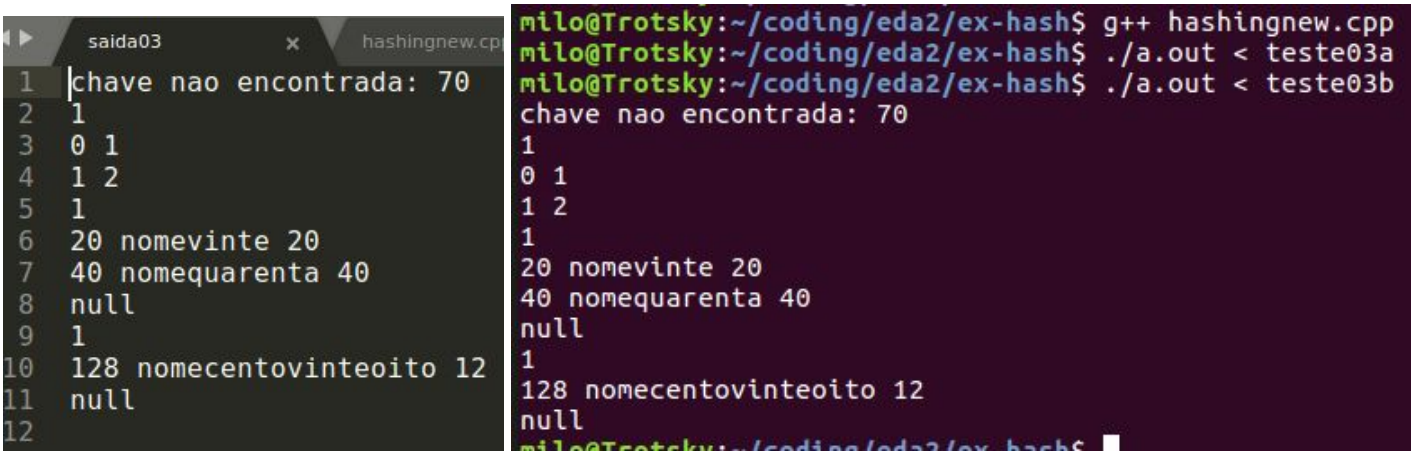
saida01: OK



saida02: OK

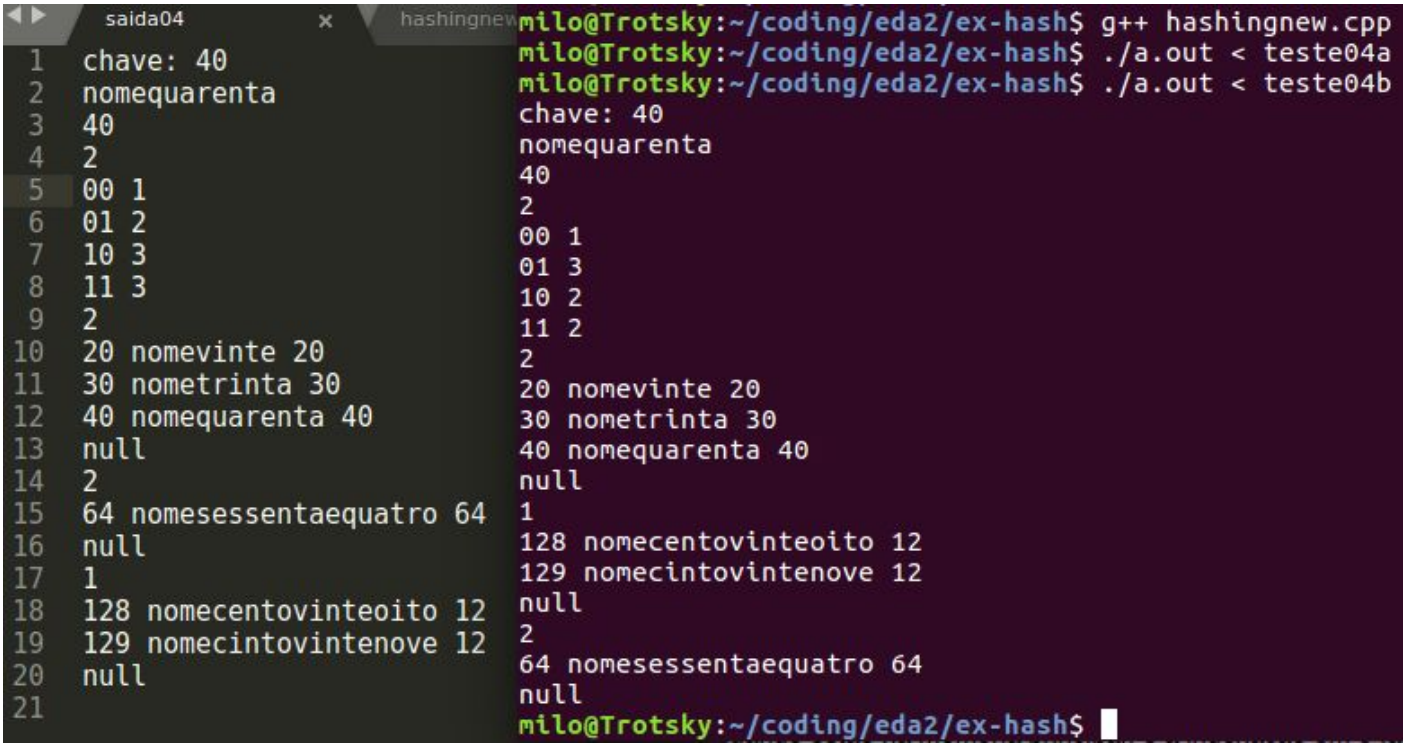


saida03: OK



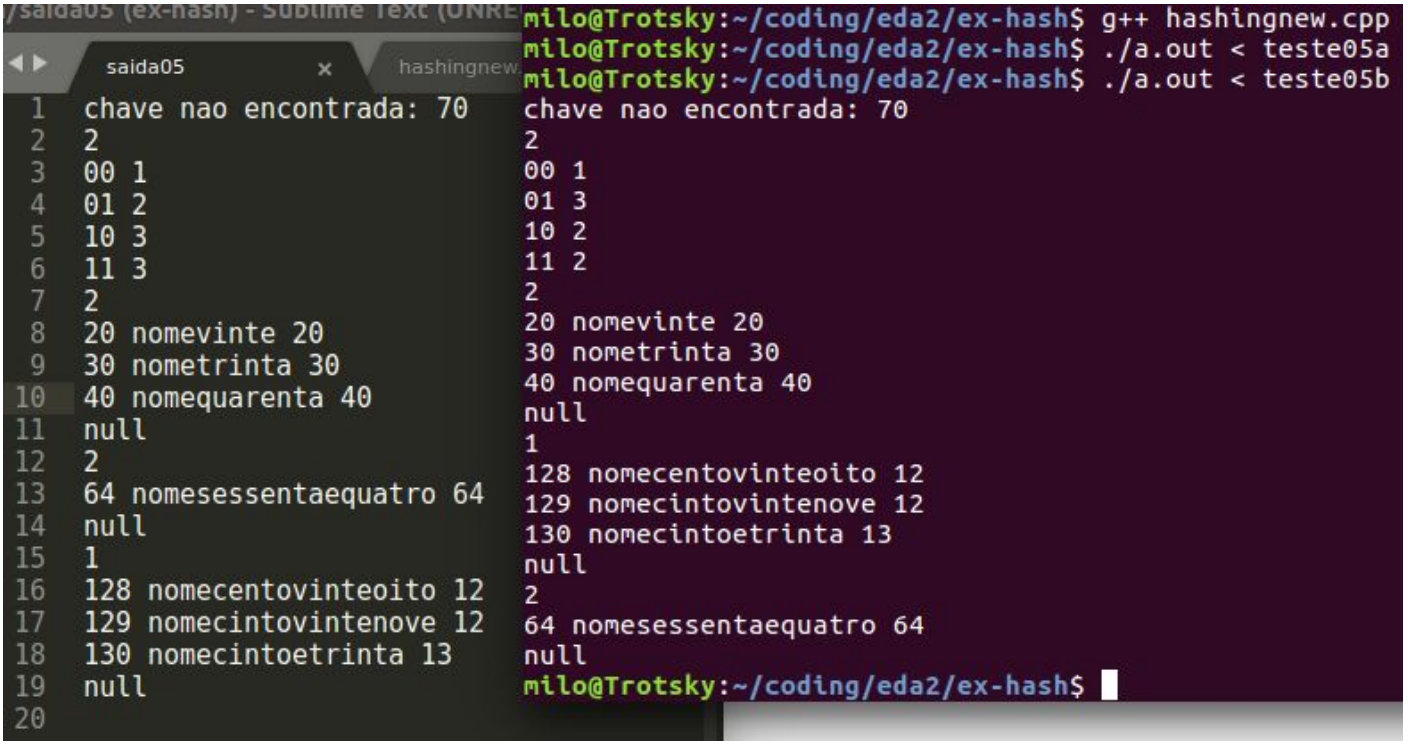
saida04:

Única diferença é a ordem das páginas, pois meu programa as armazena em ordem de criação. Porém note que os apontadores da tabela apontam para as páginas corretas e portanto encontram os registros.



saida05:

Mesma situação da saida04, a única diferença é a ordem que as páginas estão armazenadas na memória.



As saídas 06 a 10 não foram consideradas pois o motivo que elas estavam incorretas é mais complexo (um problema na função splitPage, que nem sempre salva as novas páginas no final do vector corretamente, dependendo do caso de teste; o erro não chegou a afetar as entradas 01 a 05), porém o professor pediu apenas alterações simples.