Defragment Heap File with Variable Length Records

Heap File is a doubly linked list of data pages.

Structure of Data Page

Each data page stores data from beginning and the directory structure from the end.

Last four bytes in the page represent an integer that tell us the number of entries in the directory.

Directory entries are stored after this.

Each slot in directory contains following four attrivutes.

- Record identifier (provided by the user)
- Record size
- Start location of the record in the page
- Is the record present or deleted

Assume that first three attributes take four bytes each and the last attribute takes one byte.

Please note that if the page has size 'P' then we number the bytes in the page from 1 to P. In other words, address of the first byte in the page will be 1.

Input will be of the following type

1: Create new data file

2: Insert

3: Delete

4: Display file structure

5: Defragment a data page

6: Defragment the whole Heap File

-1: Exit the program

Create new data file

Provide size of data page in bytes

Provide initial number of pages

Insert

Provide record length

Provide record identifier

Delete

Provide record identifier

Display file structure

How many pages does the page have?

Contents of each page

List the whole directory structure of the page (both present and deleted record list)

Insertion Logic:

Find the first page that can accommodate the given record. Within the page, store the record in the first fit slot. If none of the existing pages can accommodate the given record then append a new page at the end of the linked list.

Deletion logic

Just mark the record as deleted in the page. If there is an empty wasted space between current record being deleted and the next record, reclaim that empty space. (This is exactly what we have already discussed in the class.)

Consider the given input file.

Lines 1 through 3 ask you to create a new heap file with two data pages and each page has 200 bytes. Remember that last four bytes will be reserved for directory entry count. Each directory entry takes 13 bytes.

Lines 4 through 6 ask you to insert record 1001 of size 40 bytes. It will get inserted in page 1. Start offset will be 1.

Lines 7 through 9 ask you to insert record 1002 of size 160 bytes. It will get inserted in page 2. Start offset will be 1.

Lines 10 through 12 ask you to insert record 1003 of size 100 bytes. It will get inserted in page 1. Start offset will be 41.

Lines 13 through 15 ask you to insert record 1004 of size 10 bytes. It will get inserted in page 1. Start offset will be 141.

Lines 16 and 17 ask you to delete record 1003. We will simply mark it as deleted.

Lines 18 through 20 ask you to insert record 1005 of size 30 bytes. It will get inserted in page 1. Start offset will be 41.

Lines 21 and 22 ask you to defragment page 1. You sort the records in the ascending order of their size.

Assume that you borrow a new temporary empty data page and you go on copying records in the ascending order of size to it. Afterwards, you overwrite the current page with the borrowed temporary page. After defragmenting the page, following will be the pairs of record ids and corresponding start locations in the page.

1004 1

1005 11

1001 41

Directory entry for record 1004 will be stored in location range 196 to 184 (13 bytes).

Similarly directory entries for other records will follow.

Line 23 asks you to defragment the whole file. You should defragment each data page individually.

Your code should read from standard input and write to the standard output.

Submit single file <roll_number>_defrag.cpp