



The University of New Mexico
CS 241—Data Organization: Project 4

Linked List with Memory Reuse

Joel Castellanos

Due: Thursday, 3/12/2009 at midnight in WebCT

Problem Specification

This project is an extension of the doubly linked list of Project 3. In this project the parallel arrays of the doubly linked list are replaced with a nested structure. Additionally, a linked list of free data storage is maintained. In the next iteration, both of these linked lists will be converted to heaps. This will very closely follow how `malloc` and `free` are implemented on many systems. Like the last program, this will read a data file of quotations. This implementation supports a few commands in addition to `:delete:<sorted record number>`.

- 1) Your source code must consist of the single file: `ListDB.firstname.lastname.c`
- 2) You must compile on the `cs.unm.edu` Linux machines with `gcc` version 4.2.3 for `x86_64-linux-gnu`.
- 3) Your program must open and read the file given as a command line argument.
- 4) Each correct record of the data file will consist of a sequence of ASCII characters with codes [32,126] ending with the ASCII line feed character [10].
- 5) Your program must process the following types of records:
 - a. `name:quote`
 - b. `:remove:quoteNum`
 - c. `:query:author:author substring`
 - d. `:query:quote:quotation substring`
 - e. `:print:quotelist`
 - f. `:print:data:idx`
 - g. `:print:heaplist`
 - h. `:echo:`
- 6) If any data errors are encountered within the data file, then an error message should be printed and your program should exit. The exact error messages printed must match those given in the test cases. If you like, you may print error messages to `stderr`, **but they must also be printed to `stdout`**.

- 7) Your program must use the following structures for the doubly linked list of quotation records, and for the memory heap. **Note: You will lose points if you do not use *these structures with these names*.**

```

#define HEAP_SIZE 196000
#define MAX_QUOTES 2400
struct listRecord
{ int author; //index into heap.data[] of this record's author
  int quote; //index into heap.data[] of this record's quotation.
  int size; //size of author and quote data including \0s.
  int next; //If this record is in the start list, then next is the index into
            // list.rec[] of the alphabetically next record.
            //If this record is in the free list, then next is the index into list.rec[]
            // of the next unused record.
  int previous; //index into list.rec[] of alphabetically previous record.
};

struct linkedListOfQuotations
{ int start; //index into list.rec[] of first record
  int free; //index into list.rec[] of first unused record.
  struct listRecord rec[MAX_QUOTES];
} list;

struct heapRecord
{ int garbage; // index into heap.data[] of the start of this record's free memory.
  int next; //If this record is in the start list, then next is the index into
            // heap.rec[] of a free memory block of equal or larger size.
            //If this record is in the free list, then next is the index into
            // heap.rec[] of the next unused record.
  int size; //size (in bytes) of this record's free block.
};

struct heapStructure
{ char data[HEAP_SIZE]; //Storage for input data.
  int start; //index into heap.rec[] of record with the smallest free memory block.
  int free; //index into heap.rec[] of first unused record.
  int biggestBlock; //index into heap.rec[] of last (biggest) memory block.
  struct heapRecord rec[MAX_QUOTES];
} heap; //even though this version it is not yet a heap.

```

- 8) The above structure assumes that `sizeof(int)` is at least 4 bytes. Many machines use two-byte integers. Thus, the first lines of `main()` must be:

```

if (sizeof(int) < 4)
{ printf("Error: This program requires sizeof(int)>=4\n");
  return -1;
}

```

9) You must implement the function:

```
void initializeLinkedListOfRecords()
{ list.free = 0;    //The first free record is list.rec[0].
  list.start = -1; //The list is empty.

  //Initially, the first unused list record is record 0, then 0 gets linked to 1, 1 to 2...
  int i;
  for (i=0; i<MAX_QUOTES; i++)
  { list.rec[i].next = i+1;
  }
  list.rec[MAX_QUOTES-1].next = -1;
}
```

10) You must implement the function:

```
void initializeHeap()
{ //Initially, the heap contains a single block of memory of size HEAP_SIZE.
  heap.start = 0;    //Index into heap.rec[]
  heap.rec[heap.start].garbage = 0; //Index into heap.data[]
  heap.rec[heap.start].size = HEAP_SIZE;
  heap.rec[heap.start].next = -1;
  heap.biggestBlock = heap.start;

  //Initially, the first unused heap record is record 1, then 1 gets linked to 2, 2 to 3...
  heap.free = 1;
  int i;
  for (i=heap.free; i<MAX_QUOTES; i++)
  { heap.rec[i].next = i+1;
  }
  heap.rec[MAX_QUOTES-1].next = -1;
}
```

11) You must implement a function: `int getMemory(int size)`. Isn't that a better name than `malloc`? This function returns an index into `heap.data[]` of the start of the *smallest* memory block that is *at least as large as* `size`. If more than one memory block has the same size, then `getMemory()` must return the smallest such index. If there is no such block, then the function must return an error code (a number that cannot be an index, such as -1).

This function will need to do some book keeping:

- a. If the smallest block that is at least as large as `size`, is larger than `size`, then this function will need to:
 - i. Save the heap record's garbage index into `heap.data[]` in a temporary variable.
 - ii. Update the new garbage index by adding `size` to it.
 - iii. Subtract `size` from the record's size (`heap.rec[i].size`).

- iv. Since the heap record is now smaller, it might need to move closer to `heap.start`. In this case, and if this heap record had been the largest, then update `heap.biggestBlock`.
 - v. Return the original value of the record's garbage index saved in the temporary variable.
- b. If the smallest block that is at least as large as `size`, is equal to `size`, then this function will need to:
- i. Remove the record from the `heap.start` linked list.
 - ii. Add the record to the start of the `heap.free` linked list.
 - iii. Return the record's garbage index.

12) Implement: **`void releaseMemory(int dataIdx, int size)`**. This is our equitant of `free()`. This function, as you might guess, releases a continuous block of memory starting at `heap.data[dataIdx]` with the number of bytes equal to `size`. This function also has some bookkeeping to do:

- a. If the *end* of this block is adjacent to the *start* of a block already in the `heap.start` linked list, with index `i`, then this function must merge the two blocks into a single block:
 - i. Add `size` to `heap.rec[i].size`.
 - ii. Set `heap.rec[i].garbage = dataIdx`
 - iii. Since `heap.rec[i]` has increased its size, it might need to be moved farther down the `heap.start` list (which is sorted by size, and for blocks of the same size, sorted by garbage index).
- b. If the *start* of this block is adjacent to the *end* of a block already in the `heap.start` linked list, with index `k`, then this function must merge the two blocks into a single block:
 - i. Add `size` (`heap.rec[i].size` if it already merged in step a) to `heap.rec[k].size`.
 - ii. Since `heap.rec[k]` has increased its size, it might need to be moved farther down the `heap.start` list (which is sorted by size, and for blocks of the same size, sorted by garbage index).
- c. If the new block is not adjacent to an existing block, then insert a new heap record in the `heap.start` linked list. The new record must be inserted in the correct sorted order in the list: from smallest block size to largest. For blocks of the same size, the block with the smallest garbage index must come first. Note: this is called **memory fragmentation**. This is the only serious problem that arises with heap implementations of `malloc` and `free`. Many professional C and C++ programmers falsely believe that memory leaks that arise from using `malloc` can only be the result of not calling `free` for each `malloc`. A good allocator will attempt to find an unused area of already allocated memory to use before resorting to expanding the heap. This is what we do in `releaseMemory` by grouping adjacent blocks and in `getMemory` by always returning the smallest block in which the new data will fit. The major problem with this method that after many millions of calls to `malloc` and `free` (not uncommon in operational code) memory can become so fragmented that the blocks are too small to use. Some implementations

attempt to avoid this by as always allocating a segment by mapping entire pages. In such implementations, mapping even a single byte will use an entire page, which is usually 4096 bytes. Although this is usually quite acceptable, many architectures use large pages (up to four megabytes). The combination of this method with large pages can potentially waste vast amounts of memory. Unlike Java, C and C++ provide the programmer with direct access to pointers. Therefore, C and C++ cannot defragment the heap since this might invalidate those programmer controlled pointers.

d. If the new (or merged) block is the largest, then update **heap.biggestBlock**.

13) Each input data record must be read into a buffer using the `stdio.h` library function:

```
char *fgets(char *s, int n, FILE *stream);
```

14) *author:quote*

The string *author* will not contain the ':' character. Data records in this form must be processed as follows:

a. Replace the linefeed character at the end of the record with a '\0'.

b. Make exactly one call to **getMemory(int size)**.

c. Starting at the returned `heap.data[]` index, copy each character from the read buffer into `heap.data[]`. This can be done very efficiently using the `string.h` library function `memcpy()` (which often is implemented as a single machine instruction).

d. Replace the delimiting ':' character with a '\0' character. Both the author and the quotation must be NULL terminated.

e. Get the index of the next free record of the **linkedListOfQuotations** structure. Let this be: `newIdx`.

f. Place the returned `heap.data[]` index from the call to **getMemory(int size)** into `list.rec[newIdx].author`.

g. Calculate the `heap.data[]` index of the first character of the quotation and place in `list.rec[newIdx].quote`.

h. Inserted the new **linkedListOfQuotations** record into the **linkedListOfQuotations** structure in alphabetical order based first on the author name and, for quotations by the same author, by the quotation text.

15) Some of the data records will be quite large (for example, one of the data files is the entire play: "The Tragedy of Hamlet, Prince of Denmark", which has some long soliloquies). Therefore, it is necessary to read each record into a large buffer. Use **&heap.data[heap.rec[heap.biggestBlock].garbage]** as this buffer. This makes use of the largest part of the heap as temporary storage without calling **getMemory()** (that is something you cannot do with `malloc` and `free`). This is safe because we know we will be making at most one call to **getMemory()** before being done with the temporary storage. If the record is an author/quote pair, then we find its size and call **getMemory()**. By the rules specified in **getMemory()**, we know that the returned index will always be at the beginning of a block, and that none of the blocks overlap. Therefore, either the returned index will be to a completely separate block or it will equal

heap.biggestBlock. In the first case, copy each character from the “buffer” to the returned block. In the second case, the buffer *is* the returned block, and there is no need to move the data. Each call to `fgets` will need to know the size of this buffer. This size changes as data is read and removed, but it is already being maintained for the other use of this same space.

16) **:remove:quoteNum**

Process input records in this form as follows:

- a. Use the `atoi()` library function from `stdlib.h` to convert the `quoteNum` string to an integer. The integer after the keyword “:remove:” specifies a previously added quotation/author pair to be removed. For example, “:remove:15” specifies the removal of the 15th quotation/author pair by starting the count of 1 at the alphabetically first pair and progressing through the pairs in alphabetical order.
- b. Remove the specified record from the `linkedListOfQuotations` structure. Let `removeIdx` be the index into `list.rec[]` of the removed record.
- c. Make a single call to `releaseMemory(int idx, int size)` where `idx` is `list.rec[removeIdx].author`, and `size` is the combined size of both the author and the quotation (including both NULL termination characters).

17) **:query:author:author substring** Your program must display the author/quotation pair of every record in which the author field *contains* the specified substring. This search is *not* case-sensitive. Print the query results with the following format strings:

```

-----> Author Query: [%s]\n" //Header
"%s:%s\n\n" //Each Result
"<----- End Author Query\n\n" //Tail

```

18) **:query:quote:quotation substring** Your program must display the author/quotation pair of every record in which the quotation field contains the specified substring. This search is *not* case-sensitive. Print the query results with the following format strings:

```

-----> Quote Query: [%s]\n" //Header
"%s:%s\n\n" //Each Result
"<----- End Quote Query\n\n" //Tail

```

19) **:print:quotelist** Print the `linkedListOfQuotations` structure. Using the format string: `"%5d %5d %7d %8d %8d [%20.20s] [%10.10s]\n"`.

20) **:print:data:idx** Print each character in the heap from `data[0]` through `data[idx]`, replacing each NULL (`'\0'`) character with an underscore character, `'_'`.

21) **:print:heaplist** Print the `heapStructure` data structure.

22) **:echo:** Echo each input line to `stdout`. Using the format string: `"echo >>>[%s]\n"`.

23) All printed output must be sent to `strout` and must be in the format given in the examples below. Your output will be redirected to a file and compared to the given test output files using the Linux `diff` program.

Example 1: quoteDB-0.txt

```
1 :echo:
2 Aa:Bbb
3 :print:quotelist
4 :print:heaplist
5 :print:data:6
6 Cccc:Dddd
7 :print:quotelist
8 :print:heaplist
9 :print:data:16
10 H:I
11 :print:quotelist
12 :print:heaplist
13 :print:data:20
14 Fffffff:Gggggg
15 :print:quotelist
16 :print:heaplist
17 :print:data:34
18 :remove:2 //removes Cccc:Dddd. Not adjacent to a heap block
19 :print:quotelist
20 :print:heaplist
21 :print:data:34 //The Cccc:Dddd in the heap but not erased.
22 too:bigforthehole
23 :print:quotelist
24 :print:heaplist
25 :print:data:52
26 I:fit
27 :print:quotelist
28 :print:heaplist
29 :print:data:52
30 :remove:1 //removing Aa:Bbb cannot merge with the hole.
31 :print:quotelist
32 :print:heaplist
33 :print:data:52
34 :remove:3 //removing I:fit should merge two blocks.
35 :print:quotelist
36 :print:heaplist
37 :print:data:52
```

```

echo >>>>[Aa:Bbb
]
echo >>>>[:print:quotelist
]
===== Linked List of Quotations =====
Index  Next  Previous  Author  Quote      Author      Quote
   0    -1     -1       0       3  [         Aa] [         Bbb]
echo >>>>[:print:heaplist
]
===== Heap Structure =====
Index   Next  Garbage   Size
   0    -1       7    195993

echo >>>>[:print:data:6
]
===== heap.data[] =====
Aa_Bbb_

```

Figure 1-2: Output from quoteDB-0.txt after processing input records 1 through 5

The `:echo:` command on input record 1 is not echoed because `echo` was off when line 1 was read.

The closing square bracket, `]` after each echoed input record is printed on the next line because the `echo` prints the linefeed character at the end of each input record.

The linked list structure and the data array after reading `Aa:Bbb` is the same as it was in project 3:

- The input record is added to `list.rec[0]` (Index = 0).
- The structure is the only record in the list. Thus, `next` and `previous` are both -1.
- The author string starts at `heap.data[0]` and the quotation starts at `heap.data[3]`.
These first 6 characters are shown with the command: `print:data:6`. As specified in the specifications, for output, each `'\0'` character in `heap.data[]` is replaced with `'_'`.

The heap structure after reading `Aa:Bbb` is one large block starting at `heap.data[7]`, right after the `'\0'` character at the end of `Aa:Bbb`.

```

echo >>>>[Cccc:Dddd
]
echo >>>>[:print:quotelist
]
===== Linked List of Quotations =====
Index  Next  Previous  Author  Quote      Author      Quote
   0     1     -1       0       3  [         Aa] [         Bbb]
   1    -1       0       7      12  [        Cccc] [        Dddd]

echo >>>>[:print:heaplist
]
===== Heap Structure =====
Index   Next  Garbage   Size
   0    -1      17    195983

echo >>>>[:print:data:16
]
===== heap.data[] =====

```

```

Aa_Bbb_Cccc_Dddd_

echo >>>>[H:I
]
echo >>>>[:print:quotelist
]
===== Linked List of Quotations =====
Index  Next  Previous  Author  Quote      Author      Quote
   0    1    -1       0       3  [          Aa] [      Bbb]
   1    2     0       7      12  [          Cccc] [     Dddd]
   2   -1     1      17      19  [           H] [       I]

echo >>>>[:print:heaplist
]
===== Heap Structure =====
Index  Next  Garbage  Size
   0   -1    21     195979

echo >>>>[:print:data:20
]
===== heap.data[] =====
Aa_Bbb_Cccc_Dddd_H_I_

echo >>>>[Fffffff:Gggggg
]
echo >>>>[:print:quotelist
]
===== Linked List of Quotations =====
Index  Next  Previous  Author  Quote      Author      Quote
   0    1    -1       0       3  [          Aa] [      Bbb]
   1    3     0       7      12  [          Cccc] [     Dddd]
   3    2     1      21      28  [        Fffffff] [   Gggggg]
   2   -1     3      17      19  [           H] [       I]

echo >>>>[:print:heaplist
]
===== Heap Structure =====
Index  Next  Garbage  Size
   0   -1    35     195965

echo >>>>[:print:data:34
]
===== heap.data[] =====
Aa_Bbb_Cccc_Dddd_H_I_Fffffff_Gggggg_

```

Figure 1-3: Output from quoteDB-0.txt after processing input records 6 through 17

Up through line 17, nothing new happens. The structure of the linked list of author/quotation pairs has exactly the same next, pervious, author and quote indexes as are produced in the project 3 program. All data is packed in the beginning of `heap.data[]` and the “heap” is one big block consisting of everything after the last data character read from the input file.

```

echo >>>>[:remove:2 //removes Cccc:Dddd. Not adjacent to a heap block
]
echo >>>>[:print:quotelist
]
===== Linked List of Quotations =====
Index  Next  Previous  Author  Quote  Author  Quote
   0    3     -1      0      3  [      Aa] [      Bbb]
   3    2      0     21     28  [      Fffffff] [      Gggggg]
   2   -1      3     17     19  [      H] [      I]

echo >>>>[:print:heaplist
]
===== Heap Structure =====
Index  Next  Garbage  Size
   1    0      7      10
   0   -1     35    195965

echo >>>>[:print:data:34 //The Cccc:Dddd in the heap but not erased.
]
===== heap.data[] =====
Aa_Bbb_Cccc_Dddd_H_I_Fffffff_Gggggg_

```

Figure 1-4: Output from quoteDB-0.txt after processing input records 18 through 21

The freed heap space from removing the second record in alphabetically sorted order is not adjacent to the large block. The heap is now slightly fragmented, and contains two records. The new heap record (`heap.rec[1]`) was removed from the front of `heap.free` and placed in the `heap.start` list. It is inserted at the front of the `heap.start` list because it is the smallest block in the heap. The deleted record: **Cccc/Dddd**, contains 10 characters including the two terminating '\0' characters. Figure 1-3 shows that before the author/quote record was deleted, `link.rec[1].author` had the value 7. Therefore, the first fragment to be added to the heap starts at `heap.data[7]` and has a size of 10 bytes. Notice that the data **Cccc/Dddd** has not been erased. The data will not be overwritten until a record is added that fits into that block.

```

echo >>>>[:too:bigforthehole
]
echo >>>>[:print:quotelist
]
===== Linked List of Quotations =====
Index  Next  Previous  Author  Quote  Author  Quote
   0    3     -1      0      3  [      Aa] [      Bbb]
   3    2      0     21     28  [      Fffffff] [      Gggggg]
   2    1      3     17     19  [      H] [      I]
   1   -1      2     35     39  [      too] [bigfortheh]

echo >>>>[:print:heaplist
]
===== Heap Structure =====
Index  Next  Garbage  Size
   1    0      7      10
   0   -1     53    195947

echo >>>>[:print:data:52
]
===== heap.data[] =====
Aa_Bbb_Cccc_Dddd_H_I_Fffffff_Gggggg_too_bigforthehole_

```

Figure 1-5: Output from quoteDB-0.txt after processing input records 22 through 25

```

echo >>>>[I:fit
]
echo >>>>[:print:quotelist
]
===== Linked List of Quotations =====
Index  Next    Previous  Author   Quote      Author      Quote
   0     3        -1        0        3  [        Aa] [      Bbb]
   3     2         0        21       28  [        Fffffff] [    Gggggg]
   2     4         3        17       19  [         H] [         I]
   4     1         2         7         9  [         I] [         fit]
   1    -1         4        35       39  [         too] [bigfortheh]

echo >>>>[:print:heaplist
]
===== Heap Structure =====
Index  Next  Garbage  Size
   1     0     13       4
   0    -1     53    195947

echo >>>>[:print:data:52
]
===== heap.data[] =====
Aa Bbb I fit ddd H I Fffffff Gggggg too bigforthehole_

```

Figure 1-6: Output from quoteDB-0.txt after processing input records 26 through 29

In figure 1-5, the new record was too large for the size 10 memory block so space was allocated from the large block. The record added in figure 1-6, “I : f i t” is only 6 characters (including ‘\0’s). This does fit in the small heap block. The new author/quote record does not take up the whole heap block. It is copied into the start of the block at location 7 through 12. Therefore, the start of the remaining part of the small heap block is moved up to 13 (one character after the terminating NULL of the new quotation. The size of the small heap block is reduced from 10 to 4.

```

echo >>>>[:remove:1 //removing Aa:Bbb cannot merge with the hole.
]
echo >>>>[:print:quotelist
]
===== Linked List of Quotations =====
Index  Next    Previous  Author   Quote      Author      Quote
   3     2        -1       21      28 [        Fffffff] [   Gggggg]
   2     4         3       17      19 [           H] [         I]
   4     1         2         7       9 [           I] [        fit]
   1    -1         4       35      39 [          too] [bigfortheh]

echo >>>>[:print:heaplist
]
===== Heap Structure =====
Index  Next  Garbage  Size
   1     2     13       4
   2     0      0       7
   0    -1     53    195947

echo >>>>[:print:data:52
]
===== heap.data[] =====
Aa Bbb I fit ddd H I Fffffff Gggggg too bigforthehole_

echo >>>>[:remove:3 //removing I:fit should merge two blocks.
]
echo >>>>[:print:quotelist
]
===== Linked List of Quotations =====
Index  Next    Previous  Author   Quote      Author      Quote
   3     2        -1       21      28 [        Fffffff] [   Gggggg]
   2     1         3       17      19 [           H] [         I]
   1    -1         2       35      39 [          too] [bigfortheh]

echo >>>>[:print:heaplist
]
===== Heap Structure =====
Index  Next  Garbage  Size
   2     0      0       17
   0    -1     53    195947

echo >>>>[:print:data:52]
===== heap.data[] =====
Aa Bbb I fit ddd H I Fffffff Gggggg too bigforthehole_

```

Figure 1-7: Output from quoteDB-0.txt after processing input records 30 through 37

The remove in input record 30 creates a third fragment in the heap because it is not adjacent to either of the other two blocks. The remove in line 34 however, fills the gap between heap block 1 and 2 resulting in those two blocks being merged into a single block of size $4 + 7 + 6 = 17$.

Grading

No partial credit will be given for partly passed tests. Each test is all or nothing. Your output to stdout will be redirected to a file and it must match the posted test results character by character to receive credit. The Linux program `diff` will be used to check the results.

-100	The program does not implement the required data structures given in requirement #7.
-10x	Where x is the number of compiler warning messages.
+2x	Where x is the number of error cases passes in <code>quoteDB_ErrorScript.txt</code>
+5	Comments are sufficient, accurate, and informative
+10	The program is well organized (no spaghetti code)
+10	Adheres to the UNM CS-241 coding standards
+10	Passes <code>quoteDB-1.txt</code> . This test includes only author:quote , :print:quotelist , and :print:data:idx commands. Passing this test demonstrates that you successfully converted project 3 to use structures and made slight changes to the output format.
+10	Passes <code>quoteDB-2.txt</code> . This test includes author:quote , :print:quotelist , :print:data:idx , :query:author:author substring , and query:quote:quotation substring commands.
+5	Passes <code>quoteDB-3.txt</code> which is similar to <code>quoteDB-2.txt</code> .
+10	Passes <code>quoteDB-Hamlet.txt</code> . This test includes the full text of Shakespeare's play "The Tragedy of Hamlet, Prince of Denmark". It includes the commands: author:quote , :print:quotelist , :print:data:idx , :query:author:author substring , and :query:quote:quotation substring .
+10	Passes <code>quoteDB-4.txt</code> . This test includes the full set of commands, but is constructed such that all records added are too big to fit into any of the holes left by deleted records. This demonstrates that freed space is correctly represented and sorted in the heap structure.
+5	Passes <code>quoteDB-5.txt</code> which is similar to <code>quoteDB-4.txt</code> .
+10	Passes <code>quoteDB-6.txt</code> . This test includes the full set of commands. Some added records will fit into the holes left by previously deleted records; however, none of the deletes will create free heap blocks that can be merged with other blocks.
+10	Passes <code>quoteDB-7.txt</code> which is similar to <code>quoteDB-6.txt</code> .
+10	Passes <code>quoteDB-8.txt</code> . This test includes the full set of commands and exercise both freed memory reuse and freed block merging.
+10	Passes <code>quoteDB-9.txt</code> . Similar to <code>quoteDB-8.txt</code> except with a very large file.

125 possible points out of 100, thus, this includes up to 25 extra credit points.

Extra Challenge: Clean up after yourself! (20 points)

Implement a similar lazy defragmentation algorithm as in project 3. This, however, will be much trickier in this project with all of the reuse and the heap structure. Being able to perform defragmentation of the heap is the reason some programs include their own implementation of `malloc()` and `free()`. Defragmentation requires updating every pointer that the program holds into the defragmented memory space. Only a function that has precise information about how the program is using its pointers can do this.