Timothy Holmes (username: THOLME15)

Attempt 3

Written: Oct 11, 2020 9:25 AM - Oct 11, 2020 9:27 AM

Submission View

Released: Sep 7, 2020 10:15 PM

Question 1 1 / 1 point

How is tracking of free blocks using an explicit list implemented?

A. A table stored at the beginning of the heap gives a list of all free blocks.

B. Each block contains a pointer to the previous and next blocks.

C. Each *free* block contains a pointer to the previous and next *free* blocks, while *allocated* blocks just contain the size information.

D. Each block contains a pointer to the previous and next *free* blocks.

- ◯ Answer A.
- ◯ Answer B.
- ✔ ◯ Answer C.
- ◯ Answer D.

View Feedback

Question 2 1 / 1 point

In normal usage, and compared to implicit lists, how do explicit lists impact internal fragmentation?

A. Internal fragmentation is greater with implicit lists.
B. Internal fragmentation is greater with explicit lists.
C. Internal fragmentation stays the same.
D. This heavily depends on the definition of "normal usage."

- Answer A.
- Answer B.
- ✔ Answer C.
- Answer D.

View Feedback

Question 3 1 / 1 point

What is the strategy used by the explicit list implementation of `malloc` to choose a free block?

A. Same as implicit list: there are multiple choices (e.g., *first-fit*, *next-fit*, *best-fit*).
B. By coalescing the first few free blocks at the beginning of the list.
C. By taking the free block that is at the beginning of the list.
D. By doing a binary search within the explicit list.

- ✔ Answer A.
- Answer B.
- Answer C.

Answer D.

View Feedback

Question 4 1 / 1 point

In explicit lists of free blocks, once a block is freed, how is it inserted into the list of free blocks?

A. It is always inserted at some position that depends on the address of the block, and any other choice would be incorrect.

B. It is always inserted at the beginning of the list, and any other choice would be incorrect.

C. This is implementation dependent; it can for instance be inserted at the beginning of the list, or at some position that depends on the address of the block.

D. It is inserted in its own new list, to allow for coalescing.

Answer A.

Answer B.

✔ Answer C.

Answer D.

View Feedback

Question 5 1 / 1 point

Which statement holds?

A. The smaller the number of lists in segregated lists, the more *first-fit* search in segregated lists approximates *best-fit* in explicit lists.

B. The greater the number of lists in segregated lists, the more *best-fit* search in segregated lists approximates *first-fit* in explicit lists.

C. The greater the number of lists in segregated lists, the more *first-fit* search in segregated lists approximates *best-fit* in explicit lists.

- ○ Answer A.
- ○ Answer B.
- ✔ ○ Answer C.

▷ View Feedback

Question 6 1 / 1 point

In segregated lists of free blocks, several external lists are used. What are they?

A. They are a few lists (about 128), each of them keeping track of free blocks in a fixed size range (about 64 of them are ranges of size 1).

B. They are an unbounded number of lists, each of them keeping track of free blocks in a fixed size range (about half of them are ranges of size 1).

C. They are a few lists (about 128), each of them keeping track of free blocks in a size range, and each range can evolve during execution.

D. They are an unbounded number of lists, each of them keeping track of free blocks in a size range, and each range can evolve during execution.

- ✔ ○ Answer A.
- ○ Answer B.
- ○ Answer C.

Answer D.

View Feedback

Question 7 1 / 1 point

In the context of memory management, what is garbage?

A. Heap-allocated storage that is no longer referenced.
B. Another term for Java.
C. A heap-location that is referenced by the user, but is within a free block.
D. Another term for free memory (free blocks).

✔ Answer A.

Answer B.

Answer C.

Answer D.

View Feedback

Question 8 1 / 1 point

When a memory snapshot is seen as a graph, the nodes are memory locations. We distinguish the *heap* nodes and the *root* nodes. What are the root nodes?

A. Any variable accessible in the current function of the original C program (scope-wise).

B. Variables stored in the registers.

C. Variables stored in the `.data` and `.bss` segments.

D. Variables stored on the stack.

E. All of A, B, C, D.

F. Two or more of A, B, C, D, but not all.

- ○ Answer A.
- ○ Answer B.
- ○ Answer C.
- ○ Answer D.
- ✓ ○ Answer E.
- ○ Answer F.

▷ View Feedback

Question 9 — 1 / 1 point

What are assumptions required to implement mark-and-sweep perfectly?

A. We can find the beginning of a block given an address *within* the block.
B. We can list all the roots in the memory graph.
C. We can decide whether a memory location contains a pointer.
D. We can go from one allocated block to the next.
E. All of A, B, C, D.
F. Two or more of A, B, C, D, but not all.

- Answer A.
- Answer B.
- Answer C.
- Answer D.
- ✔ Answer E.
- Answer F.

View Feedback

Question 10 1 / 1 point

What is the strategy of mark-and-sweep garbage collection?

A. Traverse the memory graph from the roots, then delete nodes that have not been visited.

B. Increment a reference counter (marker) each time a new variable points to a zone in memory, and decrement when that variable is removed (changes value or is popped from stack). If the marker is 0, free the memory zone.

C. Start at the leaves of the memory graph, and perform a random traversal for no more than `brk` steps, then delete nodes that have not been visited.

D. Mark all the memory zones returned by `malloc`, then on garbage collection, explore the current call stack to find addresses that are not marked, and free them.

✔ ◯ Answer A.

◯ Answer B.

◯ Answer C.

◯ Answer D.

▷ View Feedback

Question 11 — 1 / 1 point

Which English description corresponds to the following type:

```
int *e[2]
```

A. e is an array[2] of pointers to ints.

B. e is a function returning an array[2] of ints.

C. e is a pointer to an array[2] of ints.

D. e is an array[2] of pointers to functions returning ints.

E. This is not a correct type.

✔ ◯ Answer A.

◯ Answer B.

◯ Answer C.

◯ Answer D.

◯ Answer E.

▷ View Feedback

Question 12 1 / 1 point

Which English description corresponds to the following type:

```
int (*(*t())()) [2]
```

A. t is a function returning an array[2] of pointers to functions returning pointers to ints.

B. t is an array[2] of pointers to functions returning pointers to functions returning ints.

C. t is a function returning a pointer to a function returning a pointer to an array[2] of ints.

D. t is a pointer to a function returning a function returning a pointer to an array[2] of ints.

E. This is not a correct type.

◯ Answer A.

◯ Answer B.

✔ ◯ Answer C.

◯ Answer D.

◯ Answer E.

▷ View Feedback

Question 13 — 1 / 1 point

Which English description corresponds to the following type:

```
int (*(*k[2])())[3]
```

A. `k` is a pointer to a function returning an array[2] of pointers to array[3] of ints.
B. `k` is an array[2] of pointers to pointers to functions returning array[3] of ints.
C. `k` is a pointer to an array[2] of pointers to functions returning array[3] of ints.
D. `k` is an array[2] of pointers to functions returning pointers to array[3] of ints.
E. This is not a correct type.

- ○ Answer A.
- ○ Answer B.
- ○ Answer C.
- ✔ ○ Answer D.
- ○ Answer E.

▷ View Feedback

Question 14 — 1 / 1 point

Which type corresponds to the following English description:

`r` is a function returning a pointer to an array[2] of pointers to functions returning pointers to pointers to ints.

A. `int **(**(r()))()[2]`

B. `int **(*(*r())[2])()`

C. `int *(**(*r))[2]()`

D. `int **(*(r()))[2]()`

- ○ Answer A.
- ✔ ○ Answer B.
- ○ Answer C.
- ○ Answer D.

▷ View Feedback

Question 15 1 / 1 point

Let's start with a bit of C fun. In C, if we have two statements `stat1` and `stat2`, then the *compound statement* `(stat1, stat2)` executes `stat1`, then `stat2`, and has the value of `stat2`.

For instance, `(v = 51, 42)` will set `v` to 51, and the compound statement itself has value 42. As another example, `x = (3, 14)` will evaluate 3 (with no effect), and set `x` to 14, as this is the value of the compound statement.

Now assume I have a standard linked list type `list_t`, and a list denoted by its head, like this:

```
typedef struct list {
  int val;
  struct list *next;
} list_t;
```

```
list_t *head;
```

What is a proper way to free this list?

A.
```
while (head) {
  list_t *p = head->next;
  free (head);
  head = p;
}
```

B.
```
void f (list_t *l) {
  if (l) f (l->next);
  free (l);
}
f (head);
```

C.
```
list_t *mf (list_t *a) {
  return a + (free (head), 0);
}
while (head && (head = mf (head->next)));
```

D.
```
for (list_t *p; head && (p = head->next, 1); head = p)
  free (head);
```

E. All of the above.

F. Some, but not all of the above.

- Answer A.
- Answer B.
- Answer C.

- Answer D.
- ✓ Answer E.
- Answer F.

View Feedback

Attempt Score:15 / 15

Overall Grade (highest attempt):15 / 15

Done