Indian Institute of Technology, Patna
Principal of Programming Languages Laboratory
Assignment-3 (Course Code: CS331)
Language: **Any Language**
Submission deadline: **17/11/2016**
Full Marks: 20+30=50

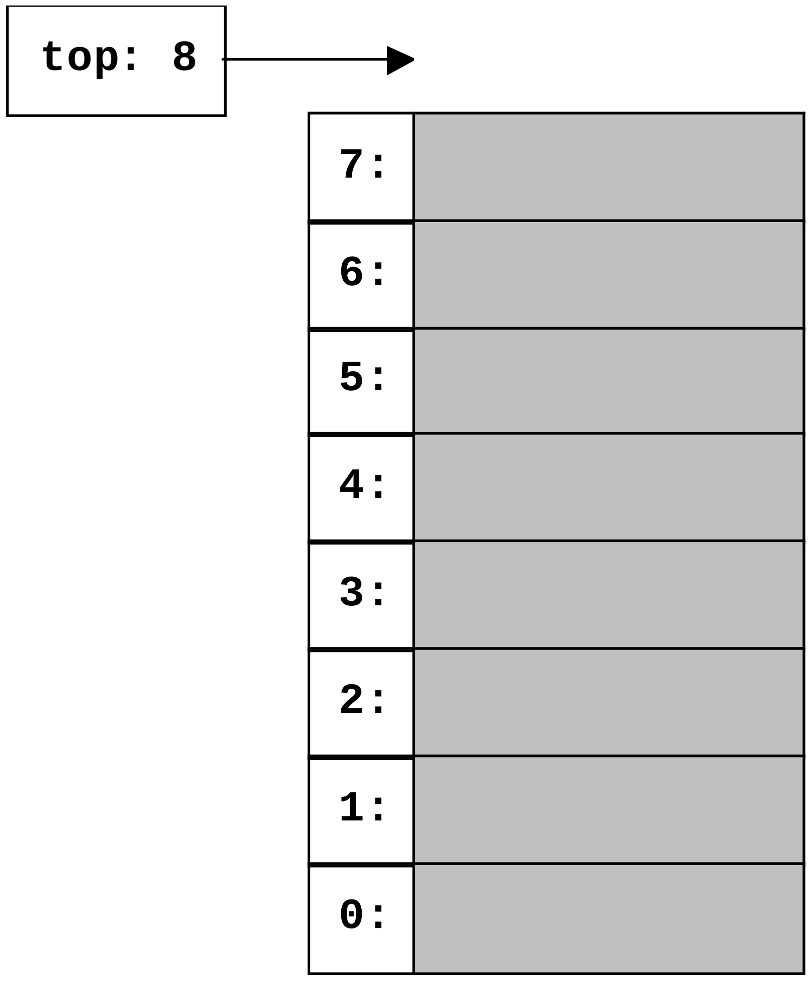
24/10/2016

- Assume that the O.S. grants each running program one or more fixed-size regions of memory (heap memory/ stack memory) for dynamic allocation. You may model these regions as arrays.
- Implement a memory manager to manage the stack and heap memory:

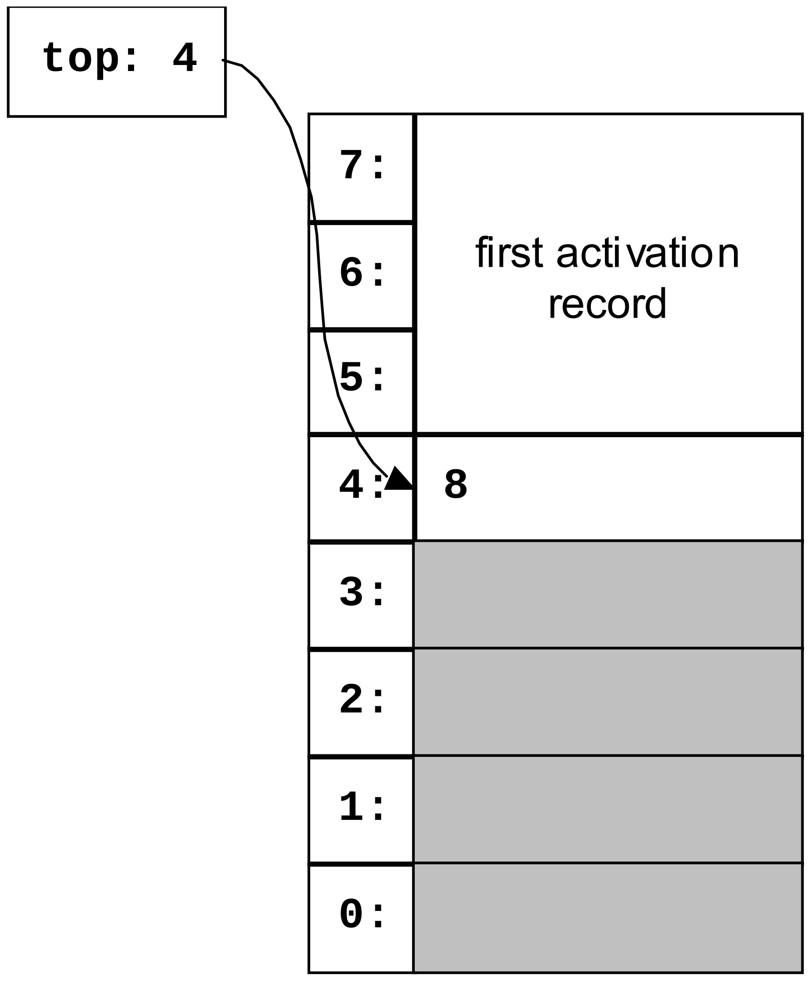
  1. For implementing memory manager for stack memory, include the following functionalities:
     (a) Push Activation Records (AR) on to the stack (instead of taking all details of AR, you may take only the size of the AR).
     (b) Pop the Activation Records (AR) from the stack.
     (c) Generate error/warning messages when required.
  2. For implementing heap memory manager:
     (a) When an allocation-request comes, allocate using the best fit, first fit and worst fit strategy (implement separately).
     (b) When a deallocation request comes, free the corresponding memory-part and add to the free-list.
     (c) Apply defragmentation, whenever required.

See the examples below for your reference.

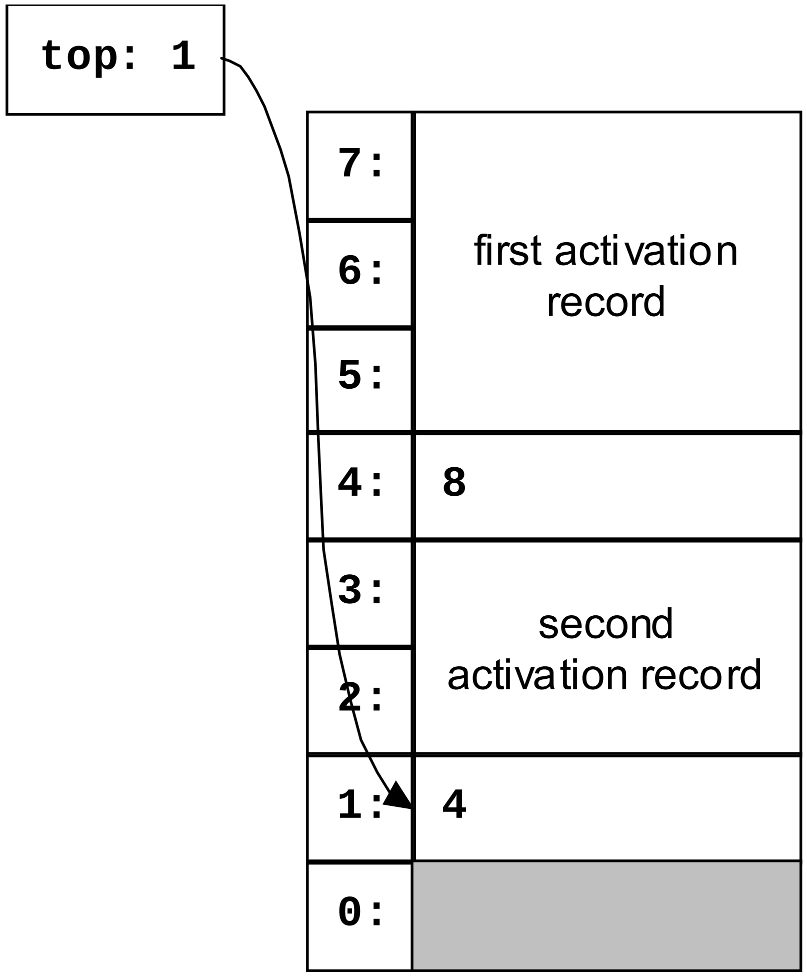
A Stack Illustration



An empty stack of 8 words. The stack will grow down, from high addresses to lower addresses. A reserved memory location (perhaps a register) records the address of the lowest allocated word.



The program calls **`m.push(3)`**, which returns 5: the address of the first of the 3 words allocated for an activation record. Memory management uses an extra word to record the previous value of **`top`**.



The program calls **`m.push(2)`**, which returns 2: the address of the first of the 2 words allocated for an activation record. The stack is now full—there is not room even for **`m.push(1)`**.

For **`m.pop()`**, just do

```
top = memory[top]
```

to return to previous configuration.

The Heap Problem

- Stack order makes implementation easy
- Not always possible: what if allocations and deallocations can come in any order?
- A *heap* is a pool of blocks of memory, with an interface for unordered runtime memory allocation and deallocation
- There are many mechanisms for this…

First Fit

- A linked list of free blocks, initially containing one big free block
- To allocate:
  - Search free list for first adequate block
  - If there is extra space in the block, return the unused portion at the upper end to the free list
  - Allocate requested portion (at the lower end)
- To free, just add to the front of the free list

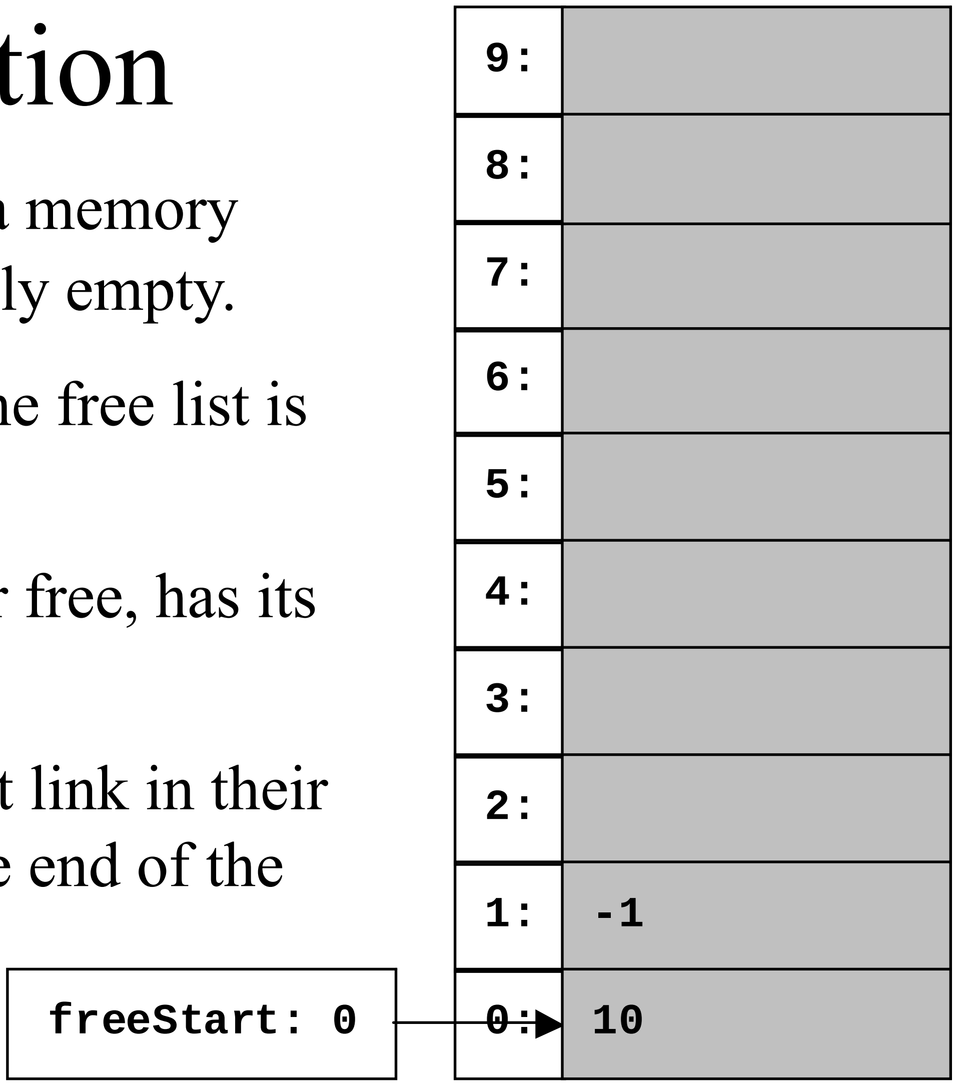
Heap Illustration

A heap manager **`m`** with a memory array of 10 words, initially empty.

The link to the head of the free list is held in **`freeStart`**.

Every block, allocated or free, has its length in its first word.

Free blocks have free-list link in their second word, or –1 at the end of the free list.

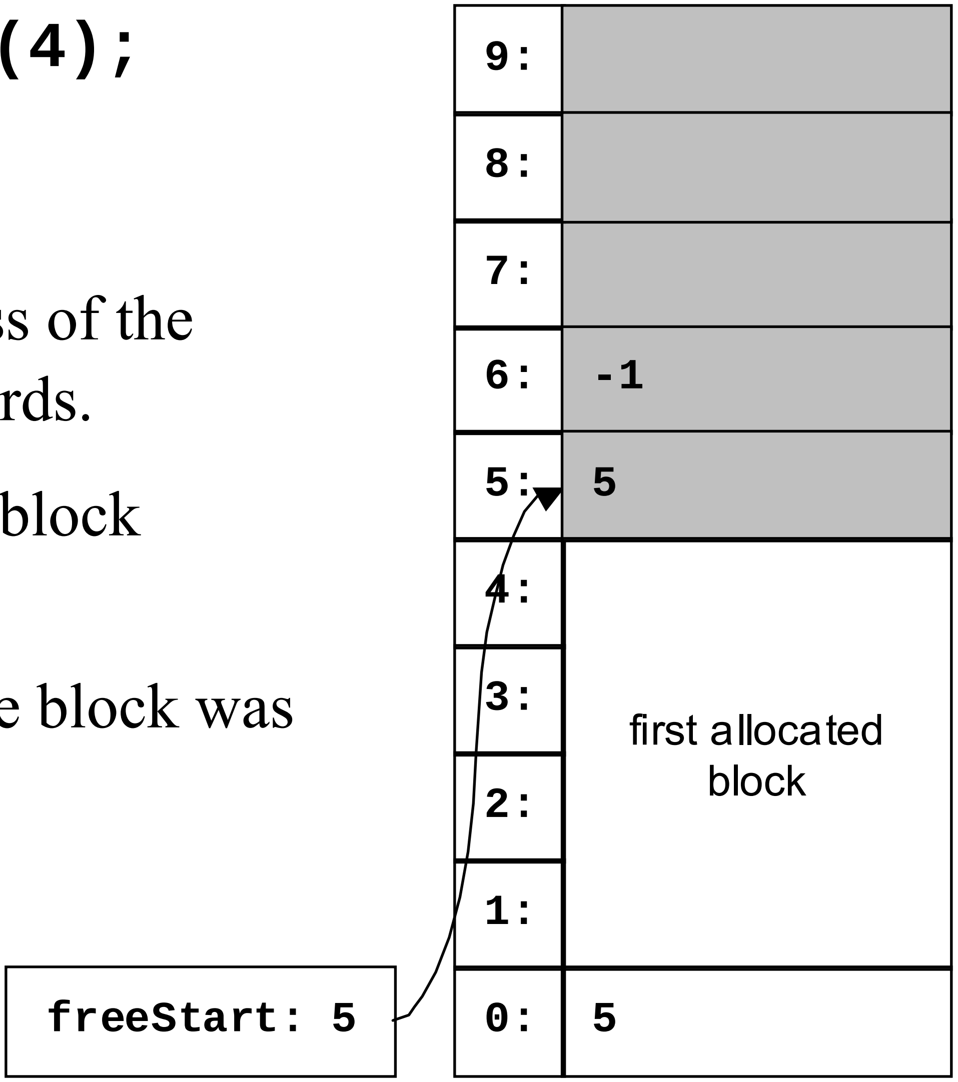


`p1=m.allocate(4);`

`p1` will be 1—the address of the first of four allocated words.

An extra word holds the block length.

Remainder of the big free block was returned to the free list.

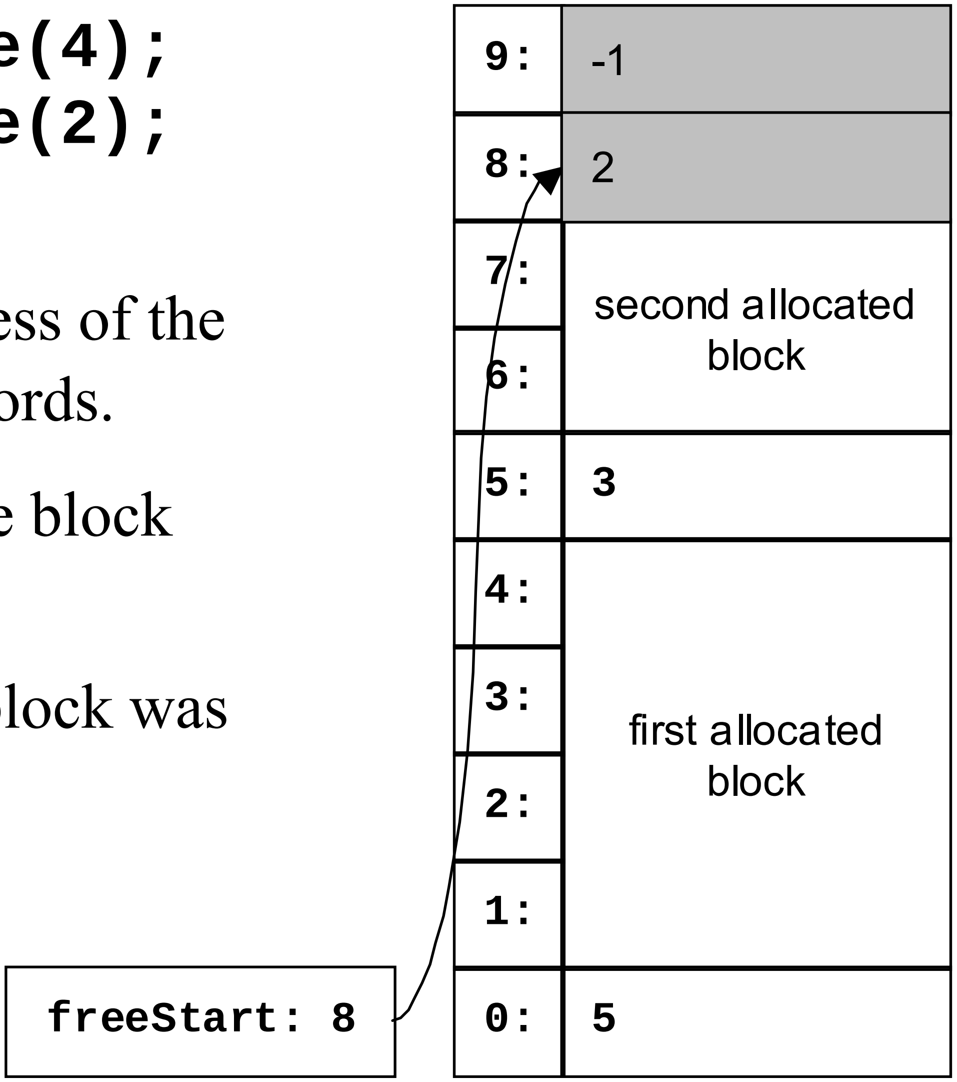


```
p1=m.allocate(4);
p2=m.allocate(2);
```

`p2` will be 6—the address of the first of two allocated words.

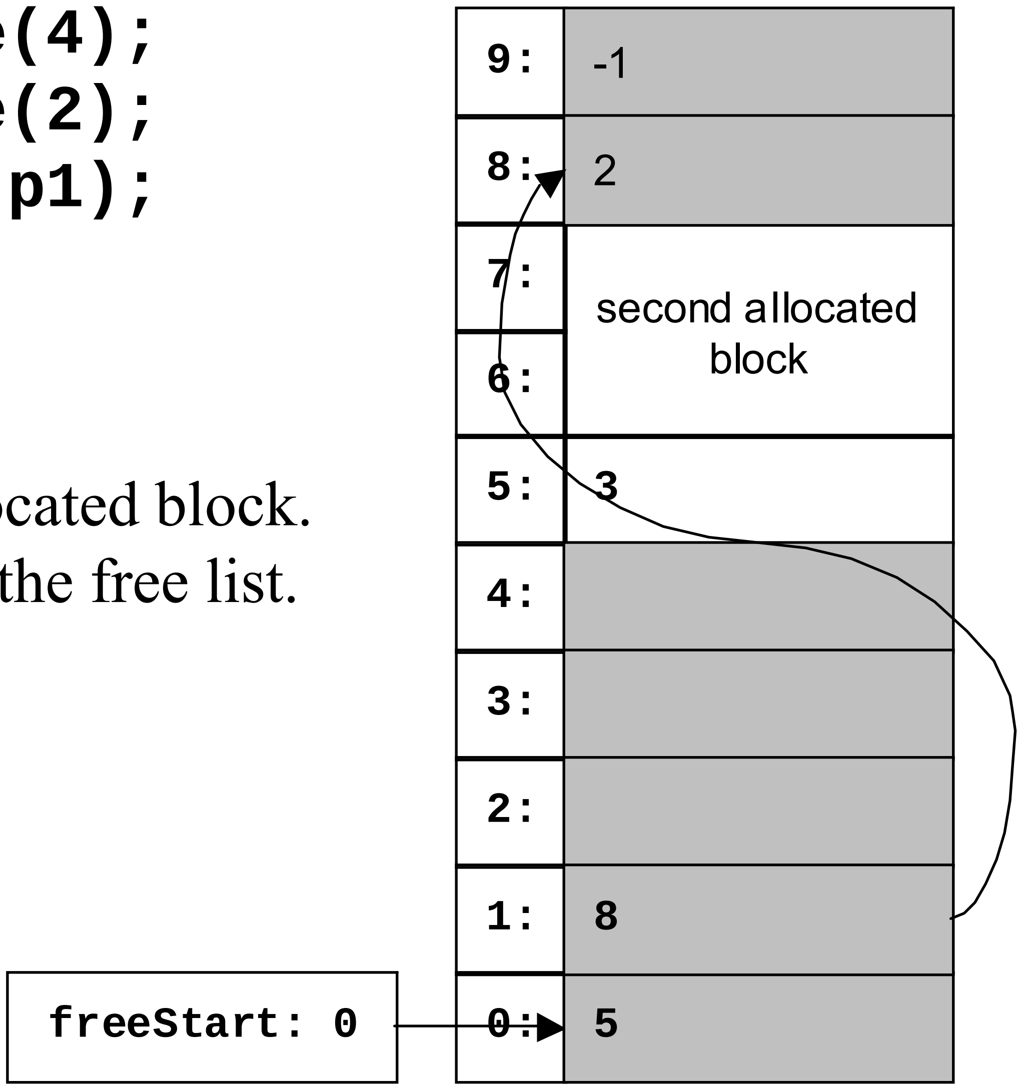
An extra word holds the block length.

Remainder of the free block was returned to the free list.



```
p1=m.allocate(4);
p2=m.allocate(2);
m.deallocate(p1);
```

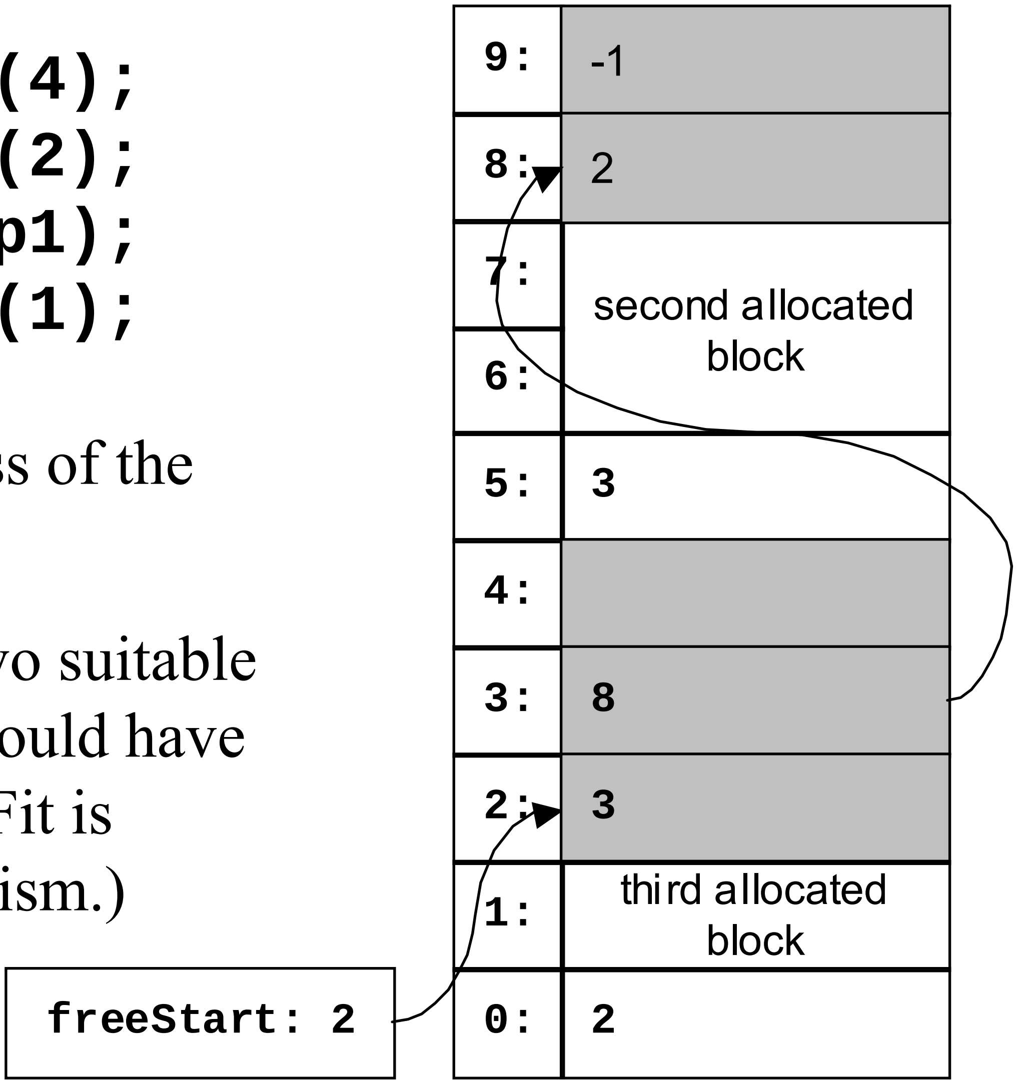
Deallocates the first allocated block.
It returns to the head of the free list.



```
p1=m.allocate(4);
p2=m.allocate(2);
m.deallocate(p1);
p3=m.allocate(1);
```

`p3` will be 1—the address of the allocated word.

Notice that there were two suitable blocks. The other one would have been an exact fit. (Best Fit is another possible mechanism.)


9:
-1
8:
2
7:
6:
second allocated block
5:
3
4:
3:
8
2:
3
1:
third allocated block
0:
2
freeStart: 2