Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Fall Term 2012

**SYSTEMS PROGRAMMING AND COMPUTER ARCHITECTURE**
**Assignment 11: Memory management**

Assigned on: **13th Dec 2012**
Due by: **20th Dec 2012**

# 1 Implicit free lists

- Determine the block sizes and header values that would result from the following sequence of `malloc` requests. Assumptions: (1) The allocator maintains a double-word alignment and uses an implicit free list with the block format from Figure 1. (2) Block sizes are rounded up to the nearest multiple of eight bytes.

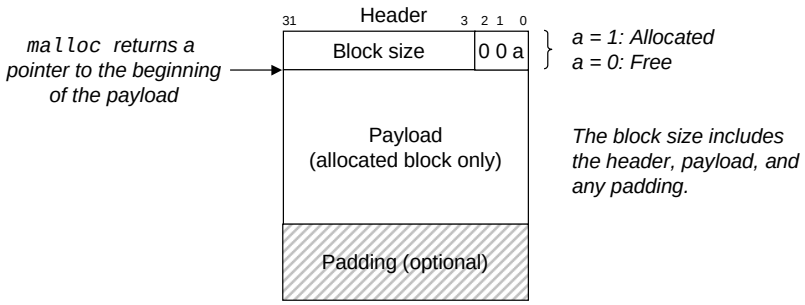



Figure 1: Format of a simple heap block for implicit free lists.

| Request | Block size (decimal bytes) | Block header (hex) |
|---|---|---|
| malloc(1) | | |
| malloc(5) | | |
| malloc(12) | | |
| malloc(13) | | |

- Determine the minimum block size for each of the following combinations of alignment requirements and block formats. Assumptions: Implicit free list, zero-sized payloads are not allowed, and headers and footers are stored in four-byte words.

| Alignment | Allocated block | Free block | Minimum block size (bytes) |
|---|---|---|---|
| Single-word | Header and footer | Header and footer | |
| Single-word | Header, but no footer | Header and footer | |
| Double-word | Header and footer | Header and footer | |
| Double-word | Header, but no footer | Header and footer | |

## 2 Dynamic memory allocation

Consider a sequence of memory allocator interactions on an IA32:

```
int   *p = (int*)malloc(2*sizeof(int));
char  *q = (char*)malloc(sizeof(char));
char  *r = (char*)malloc(sizeof(char));
float *s = (float*)malloc(2*sizeof(float));
char  *t = (char*)malloc(sizeof(char));
int   *v = (int*)malloc(sizeof(int));
/* --> (1) */
free(v);
free(r);
free(s);
/* --> (2) */
int *u = (int*)malloc(3*sizeof(int));
/* --> (3) */
```

For the following questions, make the following assumptions:

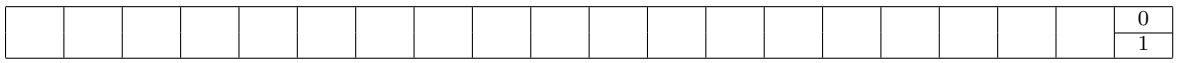
- The memory allocator uses an implicit free list to manage the space. A block in the list consists of a one-word header, the payload and possibly some additional padding. The header encodes the block size (including the header and any padding) as well as whether the block is allocated (1) or free (0).
- The memory allocator maintains double-word alignment (i.e., the address returned by the allocator is always double-word aligned). One word has 4 bytes.
- The heap has a total size of 80 bytes and starts at address `0x0806ff00`.
- The last word of the heap is occupied by a dummy header marking the end of the free list.
- The allocator uses the first-fit policy.
- The size of data types (bytes):

  `sizeof(int)=4, sizeof(char)=1, sizeof(float)=4`

**a)** Provide a picture of the heap after the above requests are processed until point `(1)` is reached. Show which bytes are allocated and which ones are free, and show the headers as appropriate. Also illustrate the pointers returned by `malloc`. You can use the following figures; each box corresponds to 1 word (4 bytes).

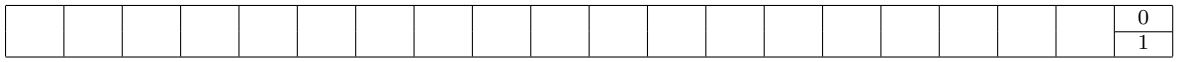
Use this space for a first draft:


Your final solution goes here:




**b)** Provide a picture of the heap after the above requests are processed until point `(2)` is reached. Assume that freed blocks are coalesced with adjacent free blocks.

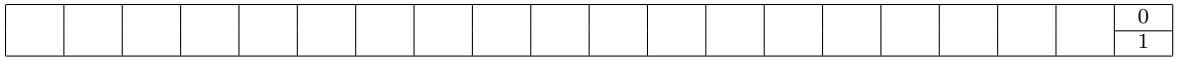
Draft:



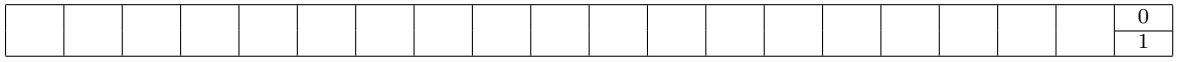

Solution:


**c)** Provide a picture of the heap after the above requests are processed until point `(3)` is reached.

Draft:




Solution:

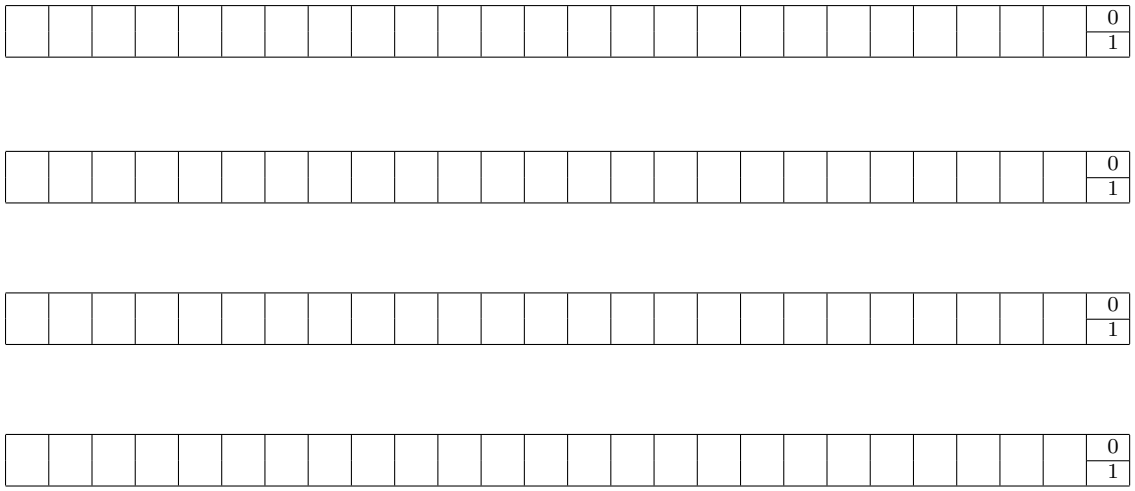



**d)** What is the internal fragmentation (total number of bytes) after the entire sequence of allocator requests has been processed? (Recall the the internal fragmentation is the sum of the differences between the sizes of the allocation blocks and their payloads.)

**e)** What is the peak memory utilization? (Recall that the peak memory utilization is the maximum of the sum of the payloads of all allocated blocks divided by the heap size.)

**f)** Give the smallest possible allocator request that (if made after all the above requests are processed) would cause external fragmentation.

**g)** Assume that the heap size is increased to 104 bytes and the memory allocator uses an implicit list with boundary tags (where a footer is placed at the end of each block which is a replicate of the corresponding header). Provide a picture of the heap after the above requests are processed until point `(3)` is reached.




# 3 Segregated free lists

This question is about a memory allocator with segregated free lists, i.e. multiple free lists where each list holds blocks within a particular size class (e.g. from 17 to 32 bytes). Assume that the free list for each size class contains same-sized blocks, each the size of the largest element of the size class.

- What are the advantages of this way to organize free blocks?
- What disadvantages exist?
- Describe a reference pattern that results in severe external fragmentation in this allocator.

# 4 Find the bugs!

Each of the following code snippets contains at least one memory-related bug. Find and explain it and propose a correct solution.

**a)**
```
int main() {
  int age;
  printf("How old are you? ");
  scanf("%d", age);
  printf("You are %d years old.\n", age);
}
```

**b)**
```
#include <stdlib.h>

void f(int n) {
  int *x = (int *) malloc(n * sizeof(int));
  // work with x
  return;
}
```

**c)**
```
#include <stdlib.h>

/* calculates y = Ax */
int *matvec(int **A, int *x, int n) {
  int i, j;
  int *y = (int *) malloc(n * sizeof(int));

  for(i = 0; i < n; i++) {
    for(j = 0; j < n; j++) {
      y[i] += A[i][j] * x[j];
    }
  }

  return y;
}
```

# Hand In Instructions

You do not need to hand this assignment in. We will publish the sample solution on the due date.