**Grading Rubric for PA1**

Functionality (85)

1. constructor 10
2. destructor 5
3. alloc 25
   1. split 10 (recursive function, or iterative, both are acceptable)
   2. Rest of the logic 15
4. free 25
   1. merge 10 (recursive function, or in-place iterative)
   2. buddy address 5
   3. Rest of the logic 10
5. LinkedList::insert() 5
6. LinkedList::remove() 10
7. Getopt() 5

Report (15)

Deduct 10 points if timing graphs are not present, deduct 5 if no analysis of bottleneck

**Deduction Criteria**

1. Check different ack(m,n) combinations. The desired results in page: <https://en.wikipedia.org/wiki/Ackermann_function> towards the end. Ackerman function working means that it will compute the correct ackerman values from the table. It should not crash when there is not enough memory.
2. If ack (3,6) works and give correct value 3 consecutive times with bbs=128 and m=64MB give full points. Also test if ack (3,8) works once to be sure their program is right.
3. Run ack(3,6) with total memory = 128KB (i.e., less than what is needed). It should not crash. If it does crash, take 10 pts off.
4. If no ack and test function work, **deduct 40** (either with segfault or without)
5. If no ackerman functions work but only the easytest(), **deduct 25**
6. If just ack(1,x) works, not ack(2,x) or ack(3,x), **deduct 20**
7. If up to ack(2,x) works, but not ack (3,x) **deduct 15**
8. If ack(3,x) works, but not ack (3,7) or (3,8) **deduct 5** points
9. Missing getop () for getting command input, **deduct 5**
10. Buddy address calculation is right only if you do it using offset. For example:

Using offset  
---------------  
Int offset = (int)((char \*) address – start\_memory);  
int buddy\_offset = offset^block\_size;  
char\* buddy = start\_memory + buddy\_offset

1. Late penalty 12**% every day, except the first 2 days when in its 6% a day**
2. The way to test whether only alloc works is the following: comment out free and run ack(3,6) with enough memory (64 MB). If it works fine, but the program breaks after using my\_free, then the problem is with my\_free. Freeproblem can be also detected when ack(3,6) works once or twice, but not multiple times. That means they cannot put the memory back to the original state.
3. Free should be have a recursive effect. To test this use the following:

Allocator->printlist(); // this should show only 1 entry in the list, the last one, the others should be all 0  
addr = allocator->alloc(1);   
allocator->printList (); // this should print everything entry has 1 block, except the last entry which should be 0;

Allocator->free(addr);  
allocator->printlist() // this again should show 1 only in the last item

The first and the third printlist () should print exactly the same output.