Practice Arena

Practice problems aimed to improve your coding skills.

- PRACTICE-02_SCAN-PRINT
- PRACTICE-03_TYPES
- LAB-PRAC-02_SCAN-PRINT
- LAB-PRAC-01
- PRACTICE-04_COND
- BONUS-PRAC-02
- LAB-PRAC-03_TYPES
- PRACTICE-05_COND-LOOPS
- LAB-PRAC-04_COND
- LAB-PRAC-05_CONDLOOPS
- PRACTICE-07_LOOPS-ARR
- LAB-PRAC-06_LOOPS
- LAB-PRAC-07_LOOPS-ARR
- LABEXAM-PRAC-01_MIDSEM
- PRACTICE-09_PTR-MAT
- LAB-PRAC-08_ARR-STR
- PRACTICE-10_MAT-FUN
- LAB-PRAC-09_PTR-MAT
- LAB-PRAC-10_MAT-FUN
- PRACTICE-11_FUN-PTR
- LAB-PRAC-11_FUN-PTR
- LAB-PRAC-12_FUN-STRUC
- LABEXAM-PRAC-02_ENDSEM
  - Meanie Numbers
  - Rotate Then Rotate Code
  - The enigma that was Enigma
  - Save the Date
  - Pretty Patterns
  - Trivial Tic-Tac-Toe
  - How Mr C reads your code
  - Malloc Mystery
- LAB-PRAC-13_STRUC-NUM
- LAB-PRAC-14_SORT-MISC

Malloc Mystery

LABEXAM-PRAC-02_ENDSEM

Malloc Mystery [70 marks]

Problem Statement

We have in this course, seen several applications of dynamic memory allocation using malloc and free (and we will see several more in the weeks to come as well). These give us the ability to return arrays from function, dynamically change the size of the array as per our need, and in general, make our life extremely simple. However, we should pause to think how hard Mr C has to work behind the scenes to make all of this work. In this problem, we will see a very very toy example to gain this appreciation.

This question will require us to simulate malloc and free calls on a toy system with only 1000 bytes of memory. As I mentioned in the 03 November lecture, not so long ago, this was actually how much memory that was available on computers. Create an integer array called MEM of length 1000. This array will act as our memory bytes. If we set MEM[i] = 0 then it will indicate that the i-th byte is not allocated to any variable. If we set MEM[i] = 1, it will mean that some variable has already been allocated this memory location. Recall that i can take valid values between 0 and 999.

The first line of the input will give you N, the number of malloc or free commands that we will give you in the next N lines. Each command will be in one of the following two forms

1. malloc
2. free

identifier will always be a single character (indicating the variable name being referred to) and block size will always be a strictly positive integer.

How to handle malloc commands

If the command is a malloc command, you have to allocate allocate a memory block from the MEM array. For example, if the command is malloc p 20, the name and size of this block will be p and 20 respectively. Store these blocks using variables of a structure Block (defined below). Please note the following details about this operation

1. malloc requires contiguous chunks of memory to be free to perform allocation. Thus, if 20 bytes are requested, malloc can work only if 20 consecutive locations in the MEM array are unassigned (i.e. are set to 0).
2. If a block of required length is free, set those memory locations in MEM as occupied (by setting them to 1) and use a Block variable to store the starting and ending indices of this block, and the name of this block (i.e. the identifier name)
3. If multiple possible consecutive 20 locations exist that are free, choose the one with the least starting index.
4. If a memory block is already allocated with name (in the current example the identifier is p), print "MEMORY LEAK" in the output. Do not free the old block. Allocate a new block if possible, using the above method and associate it with the name . The new size of M will be (in the current example, block size is 20)
5. If you are able to allocate memory this way, print "SUCCESS" (without quotes) in the output, corresponding to this command.
6. If no block of this length is available, print "FAILURE" (without quotes) in the output corresponding to this command.
7. Please note that if there is no more memory available, there cannot be a memory leak scenario since no new memory can be allocated at all. For this reason, if you get a memory leak scenario where allocation is simply not possible, do not print "MEMORY LEAK". Just print "FAILURE". Otherwise, print "MEMORY LEAK" followed by "SUCCESS" since even in the

event of memory leak, you still have to try and allocate the requested number of bytes (as Mr C also does in real life).

How to handle free commands

If the command is a free command, you have to free any memory block associated with the identifier in the command. For example if the command is free q, you have to free any memory block associated with the identifier q. Please note the following points regarding this operation

A free operation will be a success if there is a memory block associated with the name (in our example, the identifier is q). Print "SUCCESS" (without quotes) in the output corresponding to this command and also set all memory locations in the MEM array corresponding to this block to 0 indicating that they are free now.

B) A free operation will be a failure if there is no memory block associated with the name (in our example, the identifier is q). Beware that this may happen if q was earlier associated with a block but was freed earlier too so that now q is no longer associated with any block.

After handling all the commands, print the total amount of memory in use in the last line of the output.

Compulsory structure usage in your code

In your code, you should use variables of type Block where the structure is defined below:

```
struct Block{
   char name;        // Name of the memory block
   int start;        // Starting index of the memory block.
   int end;         // Ending index of the memory block.
   int valid;        // If this block is valid. (Becomes invalid once it is freed.)
};
```

Create an array of Block variables to represent the blocks you have allocated so far and the ones that have been freed so far.

```
int n;
struct Block blocks[n];
```

Be warned that not using such a structure to write your code will cause you to lose a small number of manual grading marks.

Problem-specific Words of Caution:

1. **Do not forget to submit your code**. You can submit multiple times. Your last submission will get graded.
2. Note that you have to print some message (FAILURE, SUCCESS, MEMORY LEAK) corresponding to every command we give you. The only exception are cases of malloc where memory leaks happen but allocation is still possible. In those cases, you have to print two lines on the output - one indicating the memory leak case and the other indicating that the requested malloc operation was a success. In case the requested malloc operation was a failure, do not print the warning for memory leak.
3. In addition to the above, in the last line of the output, print the total amount of memory in use after processing all the commands
4. We will not insist that you write separate functions, other than main, to solve this problem. However, solutions that do write functions to, for example, perform the malloc operation or the free operation, will get more marks.

5. We will not penalize you for extra newlines at the end of your output. However, do not have stray spaces anywhere in your output.

General Grading Policy

1. **TOTAL MARKS OF THE EXAM** 20 + 40 + 40 + 70 = 170
2. **TOTAL DURATION OF THE EXAM** 3 hours 30 minutes
3. See below for question-specific details of how partial marking would be done by the autograder in this question
4. Your submissions will be inspected by the autograder as well as a human grader
5. Human graders will (among other things) allot marks for the following

    1. Neatly structured code that uses at least one function other than the main function to process the input. The questions will usually suggest how to use functions to process the input. Submissions that ignore these suggestions and use only the main function to solve the entire problem, will lose a small fraction of marks.

    2. Proper and meaningful variable names

    3. Nice looking and consistent indentation

    4. At least a couple of comments explaining to the human grader what are you doing, especially when the steps are not obvious

    5. Comments, good indentation and meaningful variable names are very important for the human grader to understand what are you doing and why. If they cannot understand your code, do not expect them to give you (partial) marks either.

6. Solutions that indulge in hard-coding **will get a straight zero** even if they are passing some test cases. Hard-coding is a form of cheating strategy where someone write code of the form "if(input == A) printf(B)" without doing any calculations on A to obtain B. The values of A and B are either read from the evaluation/submission window or else guessed.
7. Be careful about extra/missing lines and extra/missing spaces if you do not want to lose autograder marks
8. Proportion of marks allotted to autograder (in particular, weightage to visible and hidden test cases) and human grader will be revealed when marks and grading rubrics are released
9. You are allowed to use the libraries stdio.h, math.h, string.h, stdlib.h **but not any other library**. Use of unpermitted libraries will carry a penalty. You may use any programming tools that we have discussed in lectures/tutorials or in lab questions such as arrays (1D, 2D, 3D, arrays of arrays etc), strings, loops, structures, functions, recursion, pointers, linked lists, stacks, queues, graphs, enumerations, flags, conditionals, global, static and shadowed variables.

EXAMPLE 1:
INPUT

3
malloc a 100
malloc b 100
free a

OUTPUT:
SUCCESS
SUCCESS
SUCCESS
100

EXAMPLE 2:
INPUT
3
malloc p 1000
free q
malloc q 100

OUTPUT:
SUCCESS
FAILURE
FAILURE
1000

Grading Scheme:
Total marks: **[70 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. There are 4 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

There will be no partial grading in this question. An exact match will receive full marks whereas an incomplete match will receive 0 points. Please be careful of missing/extra spaces and missing/lines (take help of visible test cases). There are 2 visible and 4 hidden test cases.

Start Solving! (/editor/practice/6253)