
Project #5 [55 points].

This is a programming project. Extra points will be given for early submission:

1 point for each day, up to 3 points.

Due date is Wednesday, December 4 (11:59 pm).

Write a C program that examines activation records in the runtime stack.

The main function should look like this:

```
int main() {
 f1();
 //f2();
 //f3();

 return 0;
}
```

You have to define 3 functions: f1(), f2(), and f3(). Run your program 3 times, calling only one of the functions on each execution.

Function `f1()` should meet the following requirements:

- It must define a local array of ***char*** values. The size must be adjustable via ***define*** macro. The default size could be 1000.
- It must also define a ***static int*** *n* that increments the number of activation records.
- It must also define a ***static long int*** *addr* that stores the starting address of the array.
- Then it must print the current activation record number, the memory address of the current array, followed by the estimated size of the current activation record as a distance (difference) between the current array address and the array address from the previous activation record.
- Finally, it must recursively call itself, if the activation record count has not exceeded 10.

Here is a sample run of `f1()`:

```
Call #1     at 000000000022FA30
AR Size     #1     is -2292272
Call #2     at 000000000022F610
AR Size     #2     is 1056
Call #3     at 000000000022F1F0
AR Size     #3     is 1056
Call #4     at 000000000022EDD0
AR Size     #4     is 1056
Call #5     at 000000000022E9B0
AR Size     #5     is 1056
Call #6     at 000000000022E590
AR Size     #6     is 1056
Call #7     at 000000000022E170
AR Size     #7     is 1056
Call #8     at 000000000022DD50
AR Size     #8     is 1056
Call #9     at 000000000022D930
AR Size     #9     is 1056
Call #10    at 000000000022D510
AR Size     #10    is 1056
-------------------------------------------------------
```

Function `f2()` should meet the following requirements:

- It must do everything `f1()` does, except this time the recursion never ends until segmentation fault.
- In addition, in each call, it must print the estimated size of the runtime stack as a product of the size of current activation record and the total count of activation records so far.

Here is a sample run of `f2()` (towards the end):

```
… … … …
Call #1959  at 0000000000036D70
AR Size     #1959  is 1056
Stack Size  #1959  is 2068704
Call #1960  at 0000000000036950
AR Size     #1960  is 1056
Stack Size  #1960  is 2069760
Call #1961  at 0000000000036530
AR Size     #1961  is 1056
Stack Size  #1961  is 2070816
Call #1962  at 0000000000036110
AR Size     #1962  is 1056
Stack Size  #1962  is 2071872
```

```
Call #1963  at 0000000000035CF0
AR Size     #1963  is 1056
Stack Size  #1963  is 2072928
Call #1964  at 00000000000358D0
AR Size     #1964  is 1056
Stack Size  #1964  is 2073984
Call #1965  at 00000000000354B0
AR Size     #1965  is 1056
Stack Size  #1965  is 2075040
Call #1966  at 0000000000035090
AR Size     #1966  is 1056
Stack Size  #1966  is 2076096
Call #1967  at 0000000000034C70
AR Size     #1967  is 1056
Stack Si
--------------------------------------------------------
```

Function `f3()` should meet the following requirements:

- It must do everything `f1()` does, except this time use *malloc* to dynamically allocate the array in the heap instead of stack.
- Also, introduce another local variable, say, ***char*** *c*, to measure the size of the current activation record.
- Make sure to **free** the array from the heap before recursive call to yourself.
- As with `f1()`, make sure the recursion ends after 10 calls.

Here is a sample run of `f3()`:

```
Call #1     at 000000000022FE17
AR Size     #1     is -2293271
Call #2     at 000000000022FDD7
AR Size     #2     is 64
Call #3     at 000000000022FD97
AR Size     #3     is 64
Call #4     at 000000000022FD57
AR Size     #4     is 64
Call #5     at 000000000022FD17
AR Size     #5     is 64
Call #6     at 000000000022FCD7
AR Size     #6     is 64
Call #7     at 000000000022FC97
AR Size     #7     is 64
Call #8     at 000000000022FC57
AR Size     #8     is 64
Call #9     at 000000000022FC17
```

```
AR Size     #9     is 64
Call #10    at 000000000022FBD7
AR Size     #10    is 64
```

Submission.

Submit the following:

- The code for entire program that includes code for all your functions;
- The complete output of your program when you execute `fn1()`;
- The part of the output of your program when you execute `fn2()` with 10 last calls before segmentation fault;
- The complete output of your program when you execute `fn3()`;

The code must be copied and pasted – no screenshots!
The output may be shown in screenshots.
You may combine everything in one Word or .pdf file or you may send your code and output separately.