Operating System and Design (19CS2106S)
Lab- 7

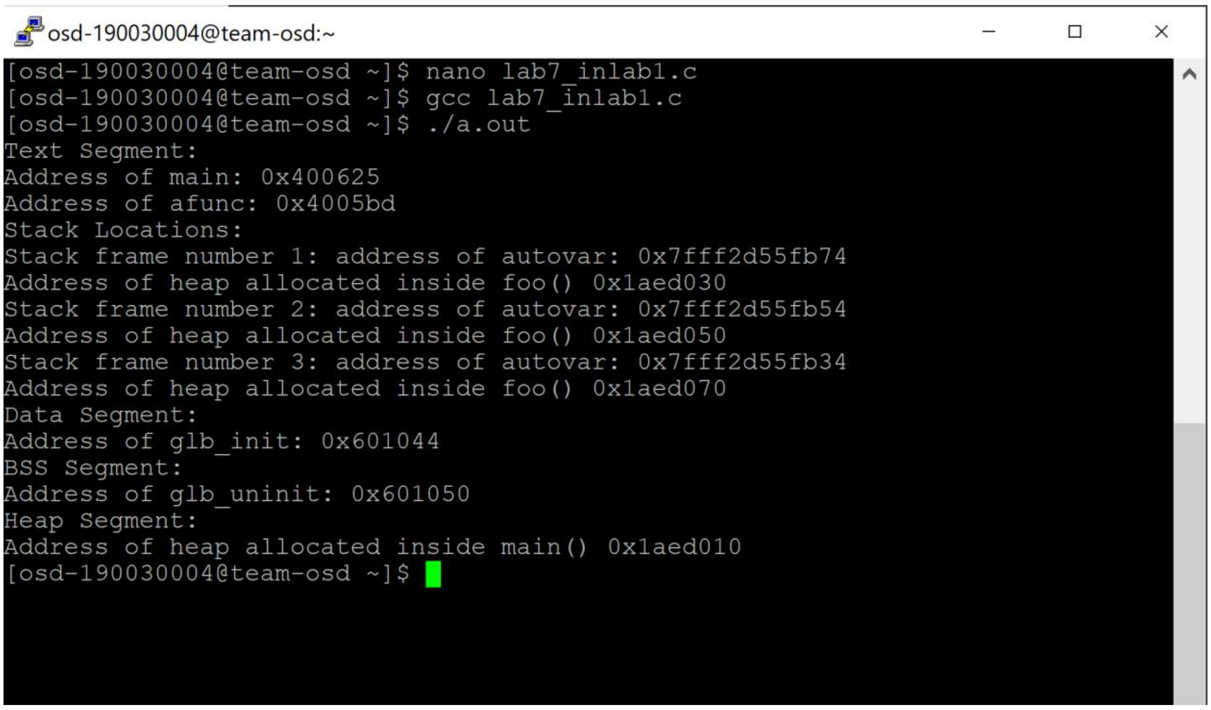
In-Lab

1. Write a program to display the address space of various segments (stack, heap, data ...etc) and show that memory address a programmer see is virtual not real.

CODE

```
#include<stdio.h>
#include<malloc.h>
int glb_uninit; /* Part of BSS Segment -- global uninitialized
variable, at runtime it is
initialized to zero */
int glb_init = 10;
/* Part of DATA Segment -- global initialized variable */
void foo(void)
{
static int num = 0;
/* stack frame count */
int autovar;
/* automatic variable/Local variable */
int *ptr_foo = (int*)malloc(sizeof(int));
if (++num == 4)
/* Creating four stack frames */
return;
printf("Stack frame number %d: address of autovar: %p\n", num, &
autovar);
printf("Address of heap allocated inside foo() %p\n",ptr_foo);
foo();
/* function call */
}
int main()
{
char *p, *b, *nb;
int *ptr_main = (int*)malloc(sizeof(int));
printf("Text Segment:\n");
printf("Address of main: %p\n", main);
printf("Address of afunc: %p\n",foo);
printf("Stack Locations:\n");
foo();
printf("Data Segment:\n");
printf("Address of glb_init: %p\n", & glb_init);
printf("BSS Segment:\n");
printf("Address of glb_uninit: %p\n", & glb_uninit);
printf("Heap Segment:\n");
printf("Address of heap allocated inside main() %p\n",ptr_main);
return 0;
}
```

OUTPUT



```
[osd-190030004@team-osd ~]$ nano lab7_inlab1.c
[osd-190030004@team-osd ~]$ gcc lab7_inlab1.c
[osd-190030004@team-osd ~]$ ./a.out
Text Segment:
Address of main: 0x400625
Address of afunc: 0x4005bd
Stack Locations:
Stack frame number 1: address of autovar: 0x7fff2d55fb74
Address of heap allocated inside foo() 0x1aed030
Stack frame number 2: address of autovar: 0x7fff2d55fb54
Address of heap allocated inside foo() 0x1aed050
Stack frame number 3: address of autovar: 0x7fff2d55fb34
Address of heap allocated inside foo() 0x1aed070
Data Segment:
Address of glb_init: 0x601044
BSS Segment:
Address of glb_uninit: 0x601050
Heap Segment:
Address of heap allocated inside main() 0x1aed010
[osd-190030004@team-osd ~]$
```

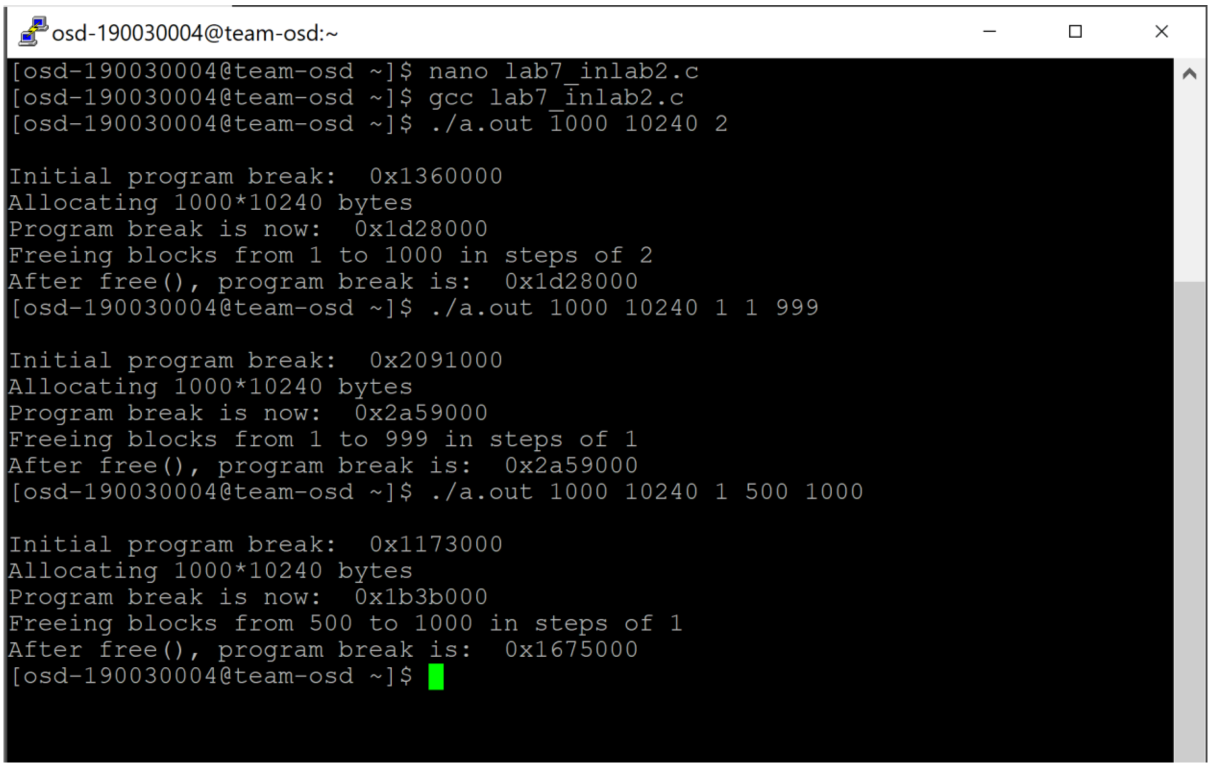
2. Develop a program to illustrate the effect of free() on the program break. This program allocates multiple blocks of memory and then frees some or all of them, depending on its (optional) command-line arguments.

CODE

```
#define MAX_ALLOCS 1000000
#include <stdio.h> /* Standard I/O functions */
#include <stdlib.h> /* Prototypes of commonly used library
functions,plus EXIT_SUCCESS and EXIT_FAILURE constants */
#include <unistd.h> /* Prototypes for many system calls */
#include <errno.h> /* Declares errno and defines error constants */
#include <string.h> /* Commonly used string-handling functions */
int main(int argc, char *argv[]){
char *ptr[MAX_ALLOCS];
int freeStep, freeMin, freeMax, blockSize, numAllocs,j;
printf("\n");
if (argc < 3 || strcmp(argv[1], "--help") == 0){
printf("%s num-allocs block-size [step [min [max]]]\n" argv[0]);
exit(5); }
numAllocs = strtol(argv[1], NULL, 10);
if (numAllocs > MAX_ALLOCS){
printf("num-allocs > %d\n", MAX_ALLOCS);
exit(5); }
blockSize = strtol(argv[2], NULL, 10);
freeStep = (argc > 3) ? strtol(argv[3], NULL, 10): 1;
freeMin = (argc > 4) ? strtol(argv[4], NULL, 10) : 1;
freeMax = (argc > 5) ? strtol(argv[5], NULL, 10) : numAllocs;
if (freeMax > numAllocs){ printf("free-max > num-allocs\n");
exit(5); }
printf("Initial program break: %10p\n", sbrk(0));
printf("Allocating %d*%d bytes\n", numAllocs, blockSize);
```

```
for (j = 0; j < numAllocs; j++) {ptr[j] = malloc(blockSize);
if (ptr[j] == NULL){ perror("malloc");
exit(5); }}
printf("Program break is now: %10p\n", sbrk(0));
printf("Freeing blocks from %d to %d in steps of %d\n",freeMin,
freeMax, freeStep);
for (j = freeMin -1;
j < freeMax;
j += freeStep)free(ptr[j]);
printf("After free(), program break is: %10p\n", sbrk(0));
exit(10);
```

OUTPUT



```
[osd-190030004@team-osd ~]$ nano lab7_inlab2.c
[osd-190030004@team-osd ~]$ gcc lab7_inlab2.c
[osd-190030004@team-osd ~]$ ./a.out 1000 10240 2

Initial program break:  0x1360000
Allocating 1000*10240 bytes
Program break is now:  0x1d28000
Freeing blocks from 1 to 1000 in steps of 2
After free(), program break is:  0x1d28000
[osd-190030004@team-osd ~]$ ./a.out 1000 10240 1 1 999

Initial program break:  0x2091000
Allocating 1000*10240 bytes
Program break is now:  0x2a59000
Freeing blocks from 1 to 999 in steps of 1
After free(), program break is:  0x2a59000
[osd-190030004@team-osd ~]$ ./a.out 1000 10240 1 500 1000

Initial program break:  0x1173000
Allocating 1000*10240 bytes
Program break is now:  0x1b3b000
Freeing blocks from 500 to 1000 in steps of 1
After free(), program break is:  0x1675000
[osd-190030004@team-osd ~]$
```

Post-Lab

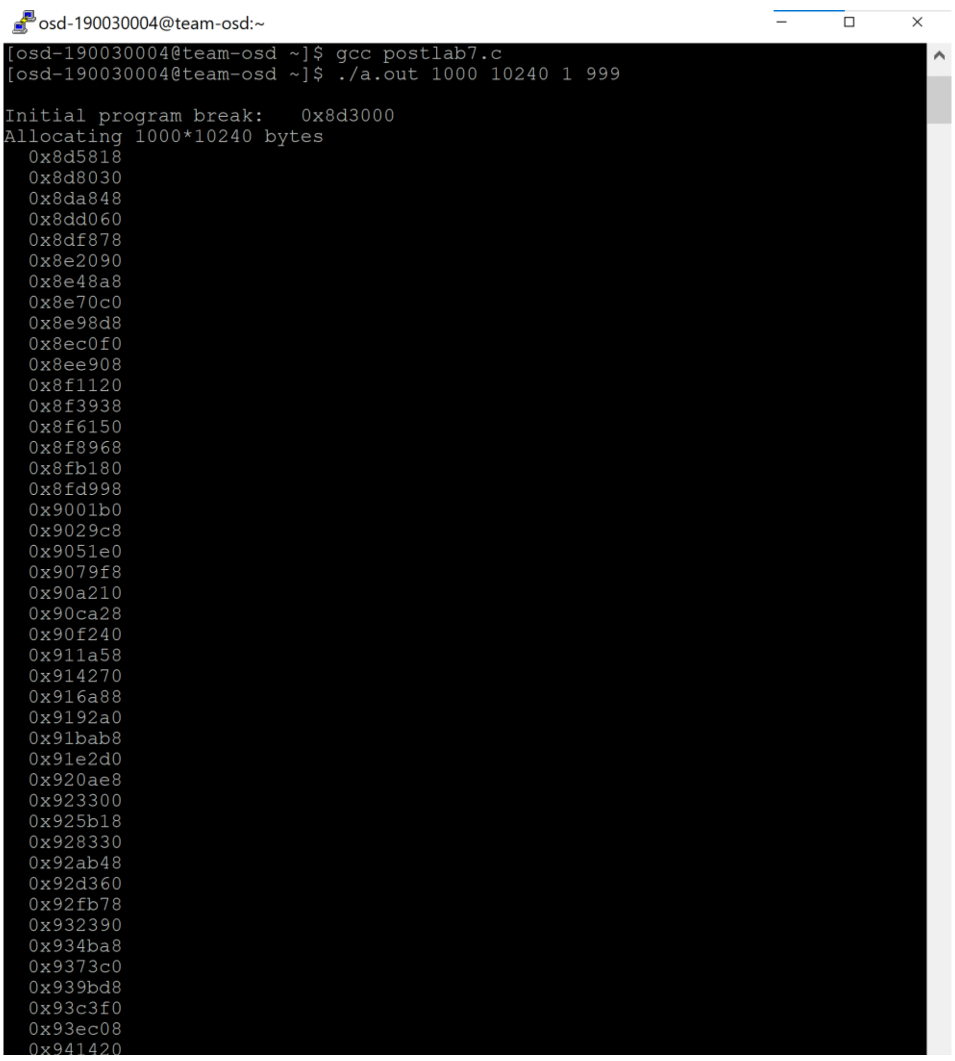
1. Write a simple memory allocator: memalloc is a simple memory allocator. Which uses your own malloc(), calloc(), realloc() and free() implemented using system calls.

CODE

```
#include <sys/types.h> /* Type definitions used by many programs */

#include <stdio.h> /* Standard I/O functions */

#include <stdlib.h> /* Prototypes of commonly used library
functions,plus EXIT_SUCCESS and EXIT_FAILURE constants */

#include <unistd.h> /* Prototypes for many system calls */

#include <errno.h> /* Declares errno and defines error constants */

#include <string.h> /* Commonly used string-handling functions */

extern char end;

void *my_malloc (size_t);

void my_free(void *);

struct blk {size_t size;

struct blk *prev;

struct blk *next;};

struct blk *first = NULL;

struct blk *last = NULL;

void *my_malloc (size_t size) {size_t required_size = size +
sizeof(struct blk);

struct blk *curr = first;

while (curr != NULL && curr->size < required_size) {curr = curr-
>next;

}if (curr == NULL) {void *new = sbrk((intptr_t) required_size);

if (new == (void *) -1) { return NULL; }

struct blk *new_blk = (struct blk *) new;

new_blk->size = required_size;
return (void *) (new_blk + 1);}

if (curr == first) { first = first->next; }

else { curr->prev->next = curr->next; }

if (curr == last) { last = last->prev; }

else {curr->next->prev = curr->prev; }if (curr->size > required_size
+ sizeof(struct blk)) {struct blk *left = (struct blk *) (((char *)
curr) + required_size);

left->size = curr->size -required_size;

curr->size = required_size;

my_free((char *) (left + 1));}return (void *) (curr + 1);}

void my_free (void *ptr) {struct blk *blk_ptr = ((struct blk *) ptr)
-1;

if (first == NULL) {first = last = blk_ptr;return;}if (blk_ptr <
first) {blk_ptr->prev = NULL;

if (((char *) blk_ptr) + blk_ptr->size == (char *) first) {blk_ptr-
>size += first->size;

blk_ptr->next = first->next;}

 else {first->prev = blk_ptr;blk_ptr->next = first;}first =
blk_ptr;return;}

if (blk_ptr > last) {if (((char *) last) + last->size == (char *)
blk_ptr) {last->size += blk_ptr->size;}

else {blk_ptr->next = NULL;

blk_ptr->prev = last;

last->next = blk_ptr;

last = blk_ptr;}

return;}

struct blk *curr = first;

while (curr < blk_ptr) {curr = curr->next;}

struct blk *before = curr->prev;

if (((char *) before) + before->size == (char *) blk_ptr) {before-
>size += blk_ptr->size;

blk_ptr = before;}
```

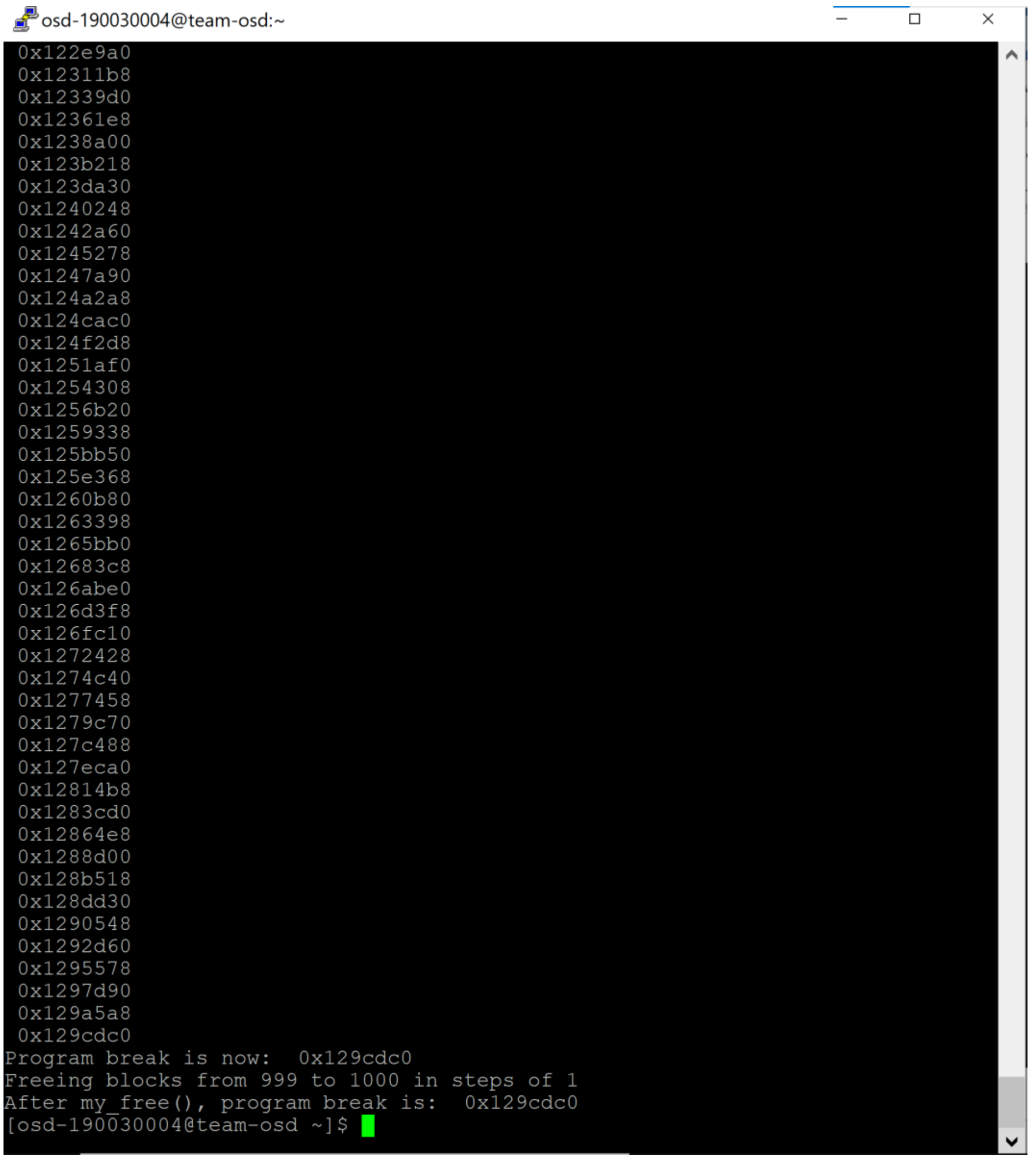
```
 else {blk_ptr->prev = before;

before->next = blk_ptr;}

if (((char *) blk_ptr) + blk_ptr->size == (char *) curr) {blk_ptr-
>size += curr->size;

blk_ptr->next = curr->next;

curr->next->prev = blk_ptr;




} else {blk_ptr->next = curr;

curr->prev = blk_ptr;}}

#define MAX_ALLOCS 1000000

int main (int argc, char *argv[]) {

/* copied from free_and_sbrk.c --licensed by Michael Kerrisk under
the GPLv3 */

char *ptr[MAX_ALLOCS];

int freeStep, freeMin, freeMax, blockSize, numAllocs, j;

printf("\n");

if (argc < 3 || strcmp(argv[1], "--help") == 0) {printf("%s num-
allocs block-size [step [min [max]]]\n", argv[0]);

perror("num-allocs block-size");}

numAllocs = strtol(argv[1], NULL, 10);

if (numAllocs > MAX_ALLOCS) {printf("num-allocs > %d\n",
MAX_ALLOCS);

perror("num-allocs");}

blockSize = strtol(argv[2], NULL, 10);

freeStep = (argc > 3) ? strtol(argv[3], NULL, 10) : 1;

freeMin = (argc > 4) ? strtol(argv[4], NULL, 10) : 1;

freeMax = (argc > 5) ? strtol(argv[5], NULL, 10): numAllocs;

if (freeMax > numAllocs) {perror("free-max > num-allocs");}

printf("Initial program break: %10p\n", sbrk(0));
printf("Allocating %d*%d bytes\n", numAllocs, blockSize);

for (j = 0; j < numAllocs; j++) {

ptr[j] = my_malloc(blockSize);

if (ptr[j] == NULL) {perror("malloc");}

printf("%10p\n", sbrk(0));}

printf("Program break is now: %10p\n", sbrk(0));

printf("Freeing blocks from %d to %d in steps of %d\n",freeMin,
freeMax, freeStep);

for (j = freeMin -1; j < freeMax; j += freeStep) {my_free(ptr[j]);}

printf("After my_free(), program break is: %10p\n", sbrk(0));

exit(EXIT_SUCCESS);}
```

OUTPUT



osd-190030004@team-osd:~

```
[osd-190030004@team-osd ~]$ gcc postlab7.c
[osd-190030004@team-osd ~]$ ./a.out 1000 10240 1 999

Initial program break:   0x8d3000
Allocating 1000*10240 bytes
  0x8d5818
  0x8d8030
  0x8da848
  0x8dd060
  0x8df878
  0x8e2090
  0x8e48a8
  0x8e70c0
  0x8e98d8
  0x8ec0f0
  0x8ee908
  0x8f1120
  0x8f3938
  0x8f6150
  0x8f8968
  0x8fb180
  0x8fd998
  0x9001b0
  0x9029c8
  0x9051e0
  0x9079f8
  0x90a210
  0x90ca28
  0x90f240
  0x911a58
  0x914270
  0x916a88
  0x9192a0
  0x91bab8
  0x91e2d0
  0x920ae8
  0x923300
  0x925b18
  0x928330
  0x92ab48
  0x92d360
  0x92fb78
  0x932390
  0x934ba8
  0x9373c0
  0x939bd8
  0x93c3f0
  0x93ec08
  0x941420
```


osd-190030004@team-osd:~

```
 0x122e9a0
 0x12311b8
 0x12339d0
 0x12361e8
 0x1238a00
 0x123b218
 0x123da30
 0x1240248
 0x1242a60
 0x1245278
 0x1247a90
 0x124a2a8
 0x124cac0
 0x124f2d8
 0x1251af0
 0x1254308
 0x1256b20
 0x1259338
 0x125bb50
 0x125e368
 0x1260b80
 0x1263398
 0x1265bb0
 0x12683c8
 0x126abe0
 0x126d3f8
 0x126fc10
 0x1272428
 0x1274c40
 0x1277458
 0x1279c70
 0x127c488
 0x127eca0
 0x12814b8
 0x1283cd0
 0x12864e8
 0x1288d00
 0x128b518
 0x128dd30
 0x1290548
 0x1292d60
 0x1295578
 0x1297d90
 0x129a5a8
 0x129cdc0
Program break is now:  0x129cdc0
Freeing blocks from 999 to 1000 in steps of 1
After my_free(), program break is:  0x129cdc0
[osd-190030004@team-osd ~]$
```