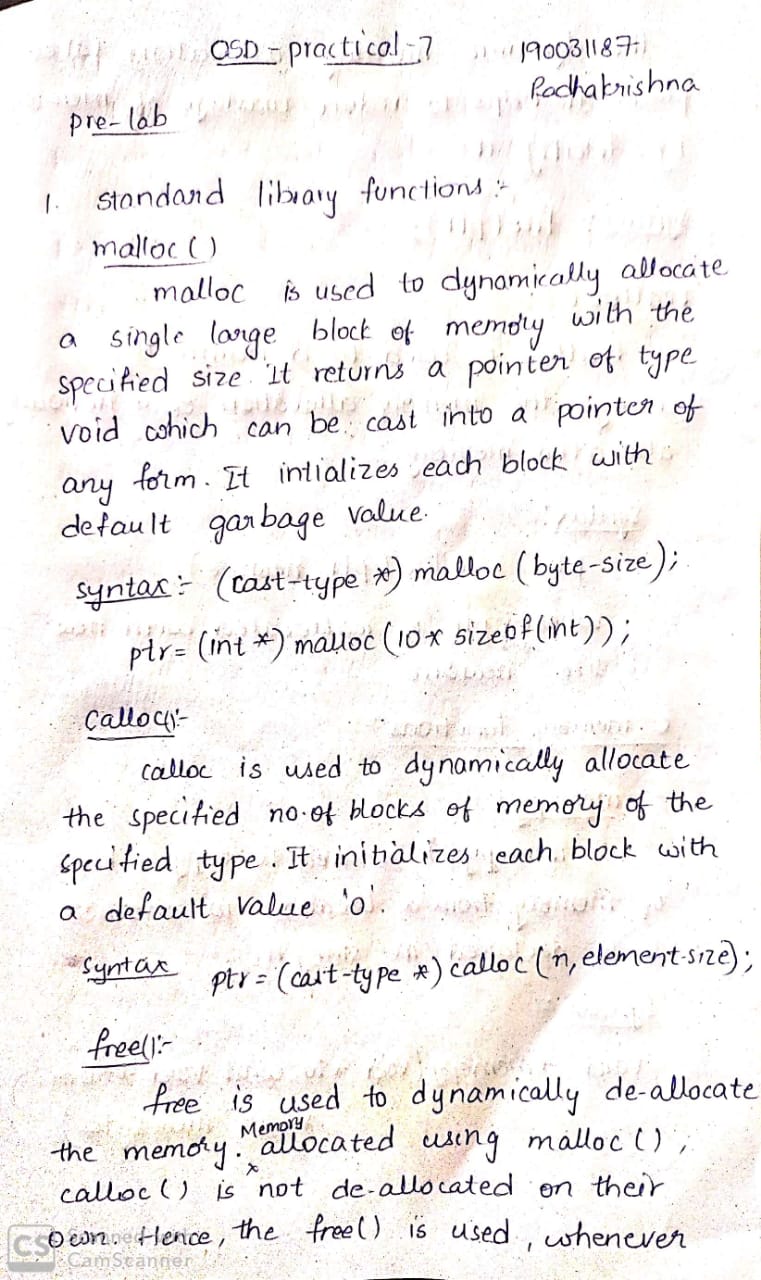
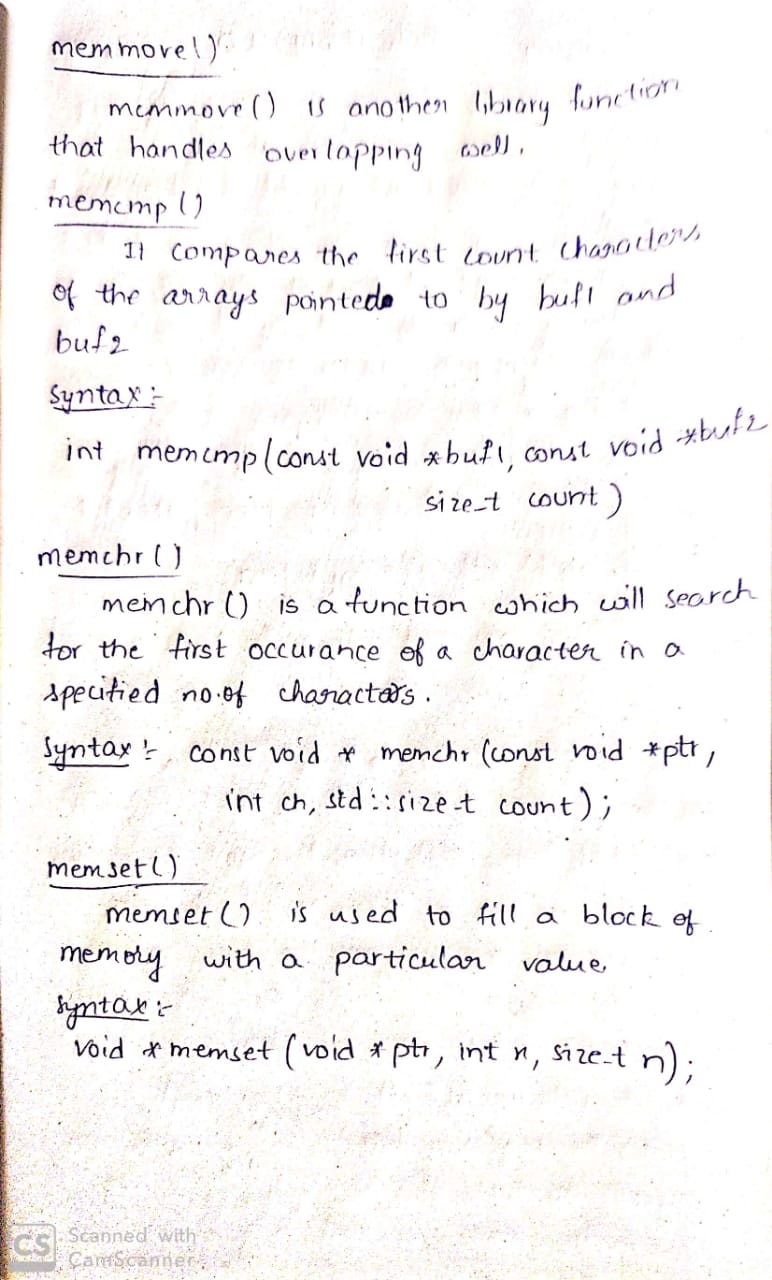
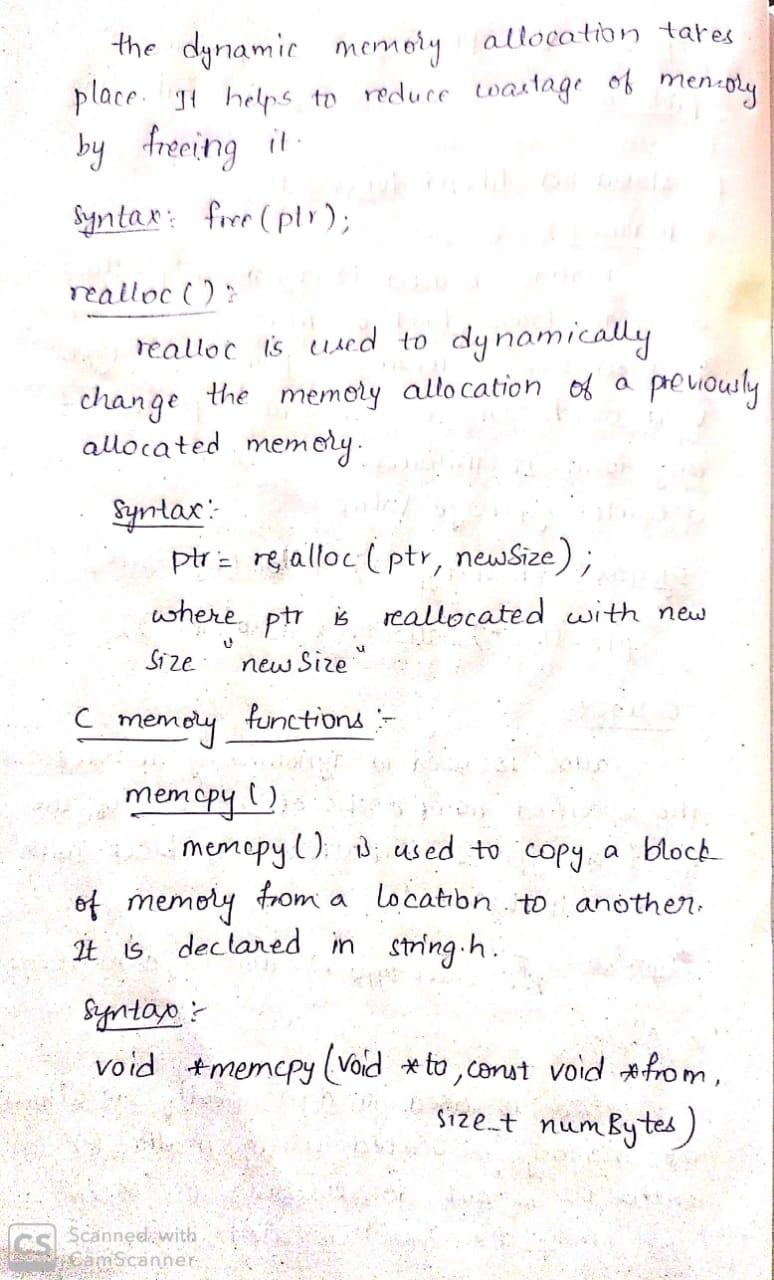
**Operating System and Design (19CS2106S)**

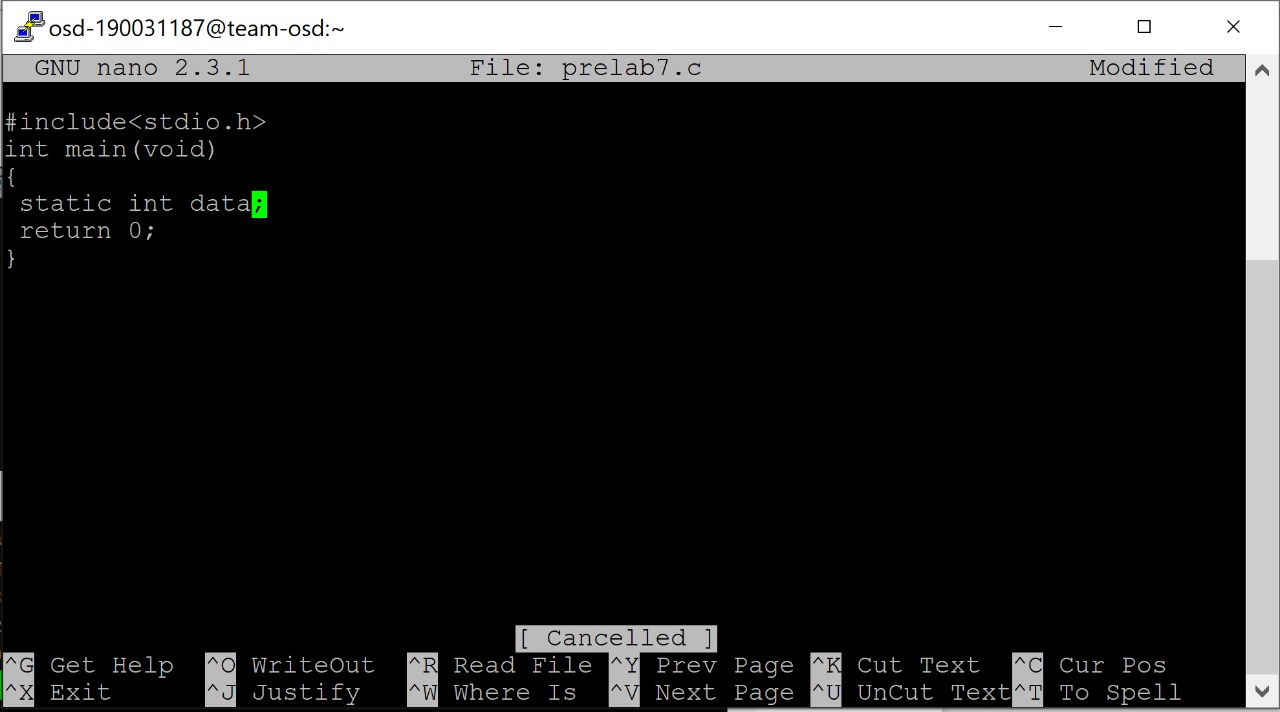
**Lab- 7**

**Pre-Lab:**

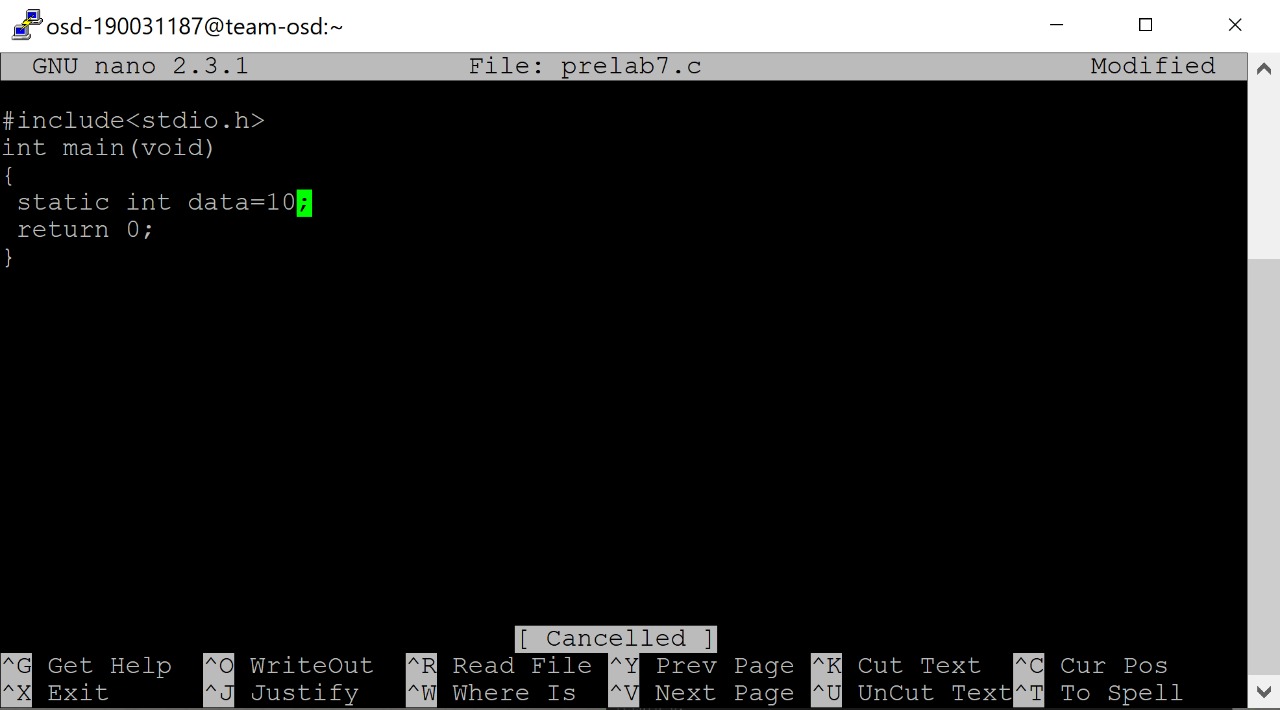
The brk and sbrk calls dynamically change the amount of space allocated for the data segment of the calling process. standard library functions: malloc(), calloc(), free() and realloc(). C memory functions: memcpy, memmove, memcmp, memchr, memset. C string functions .

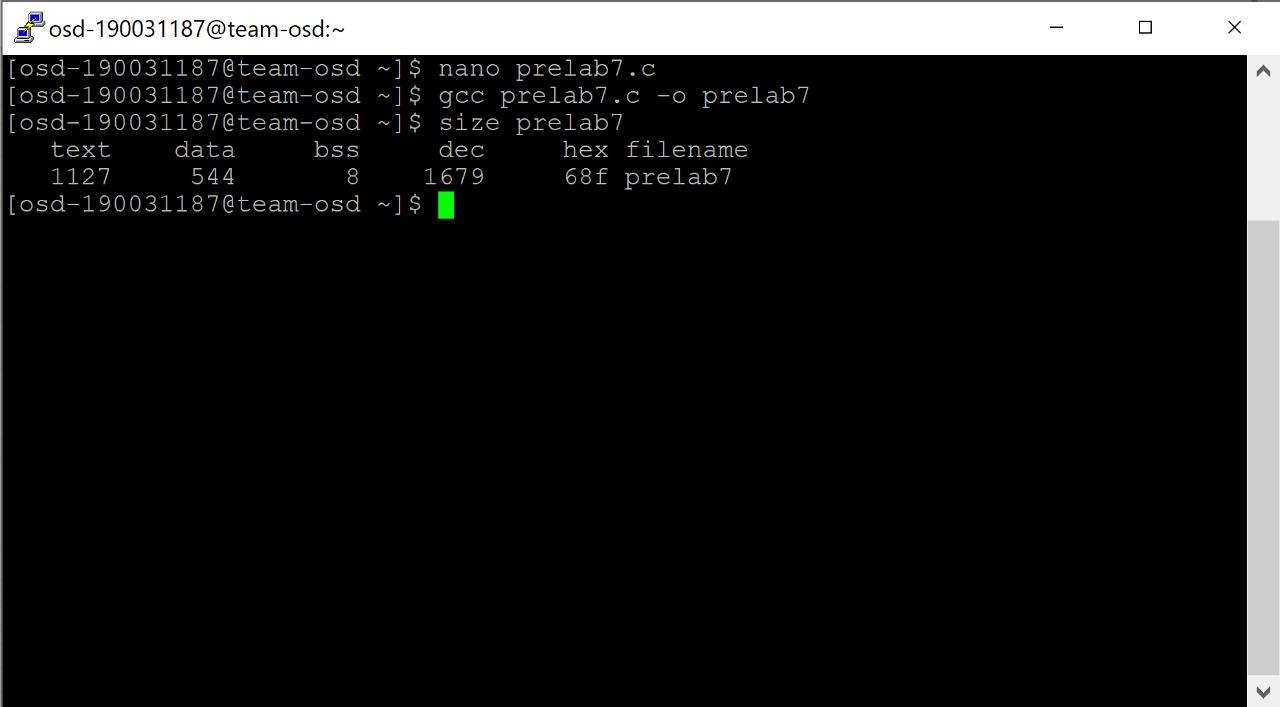
****

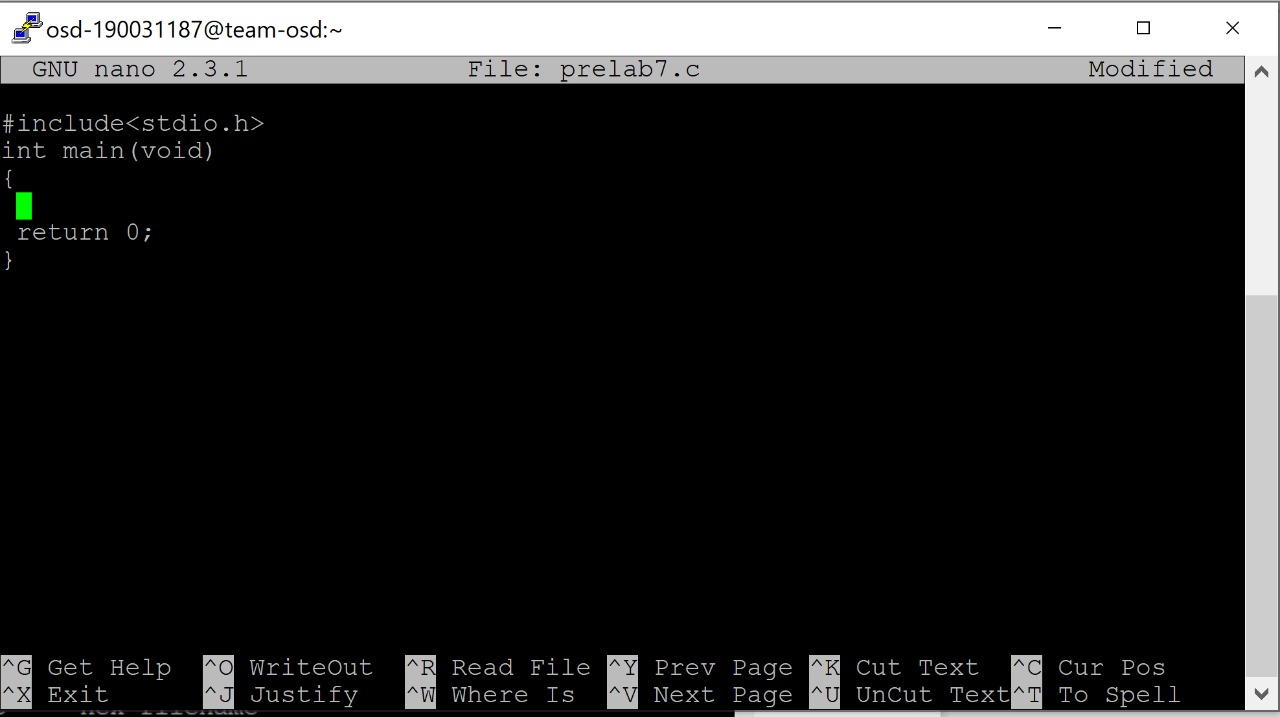
****

****

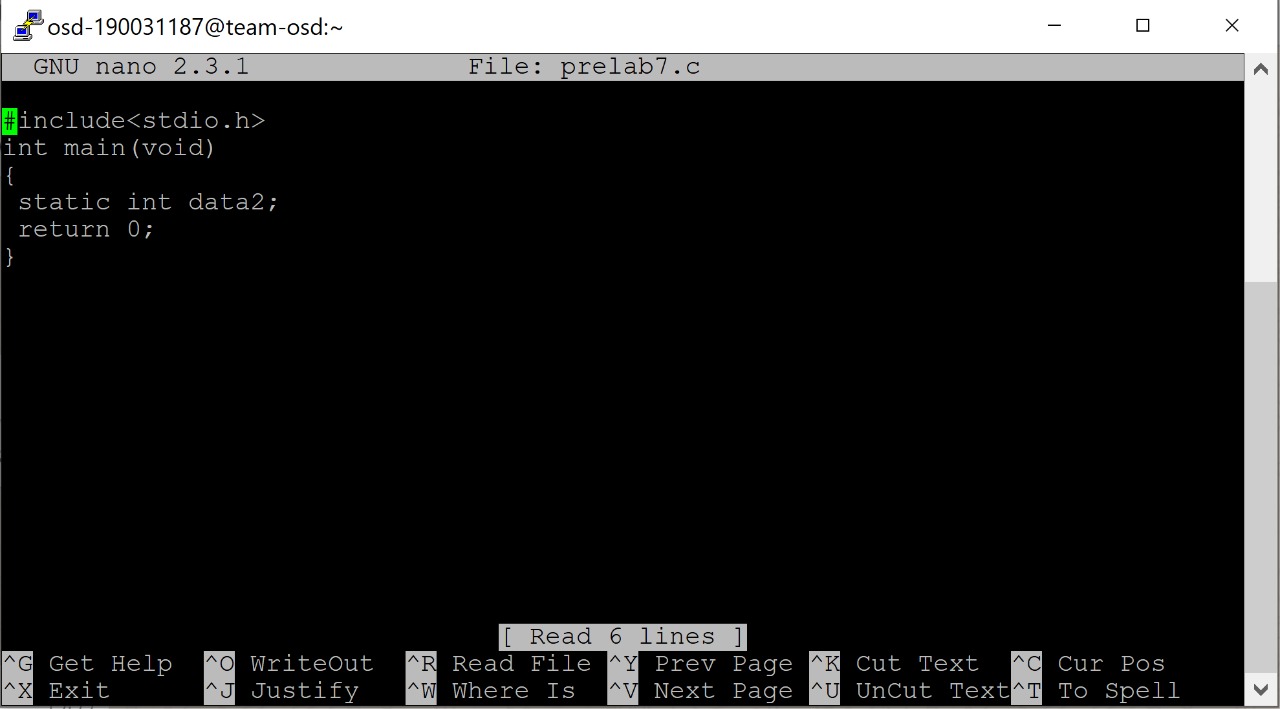
****

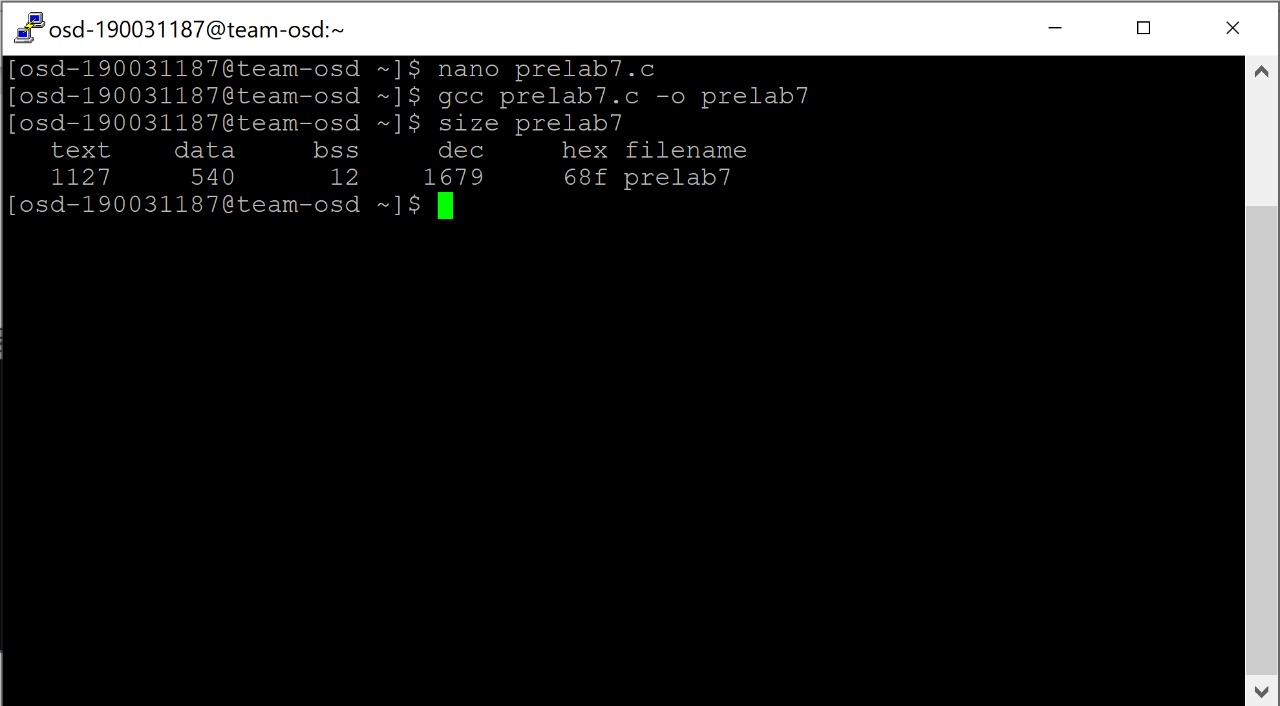
****

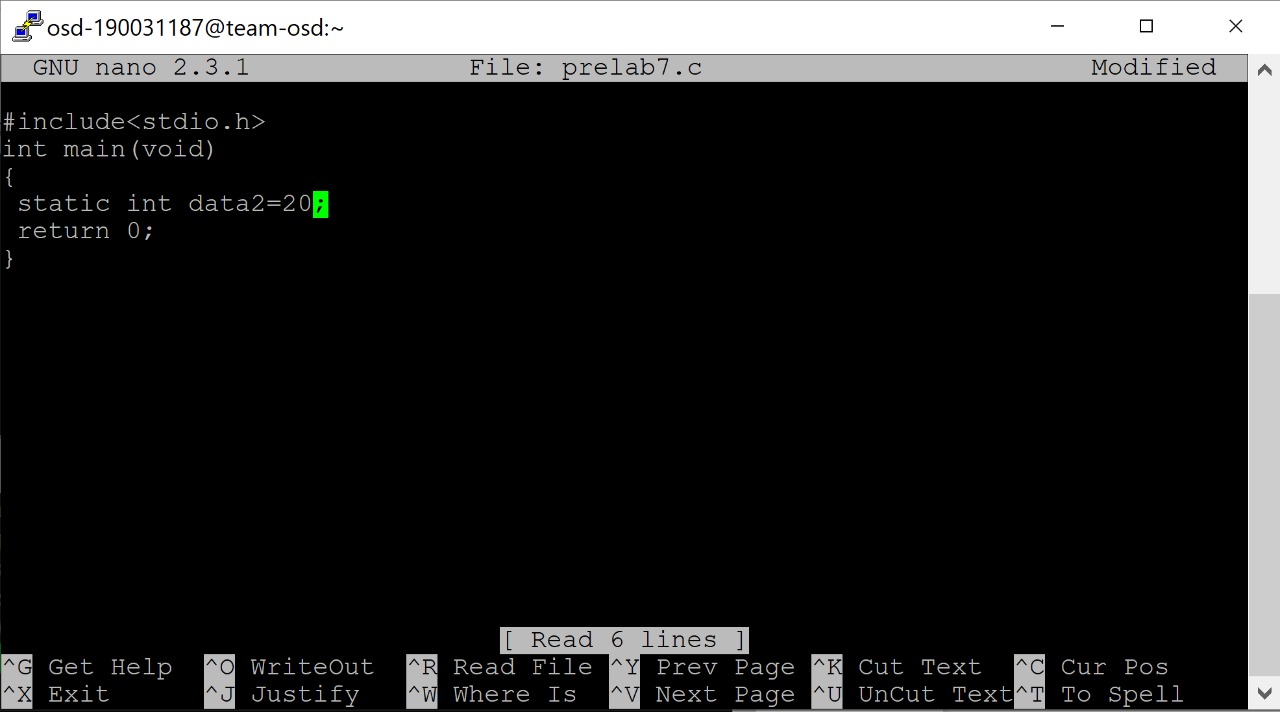
****

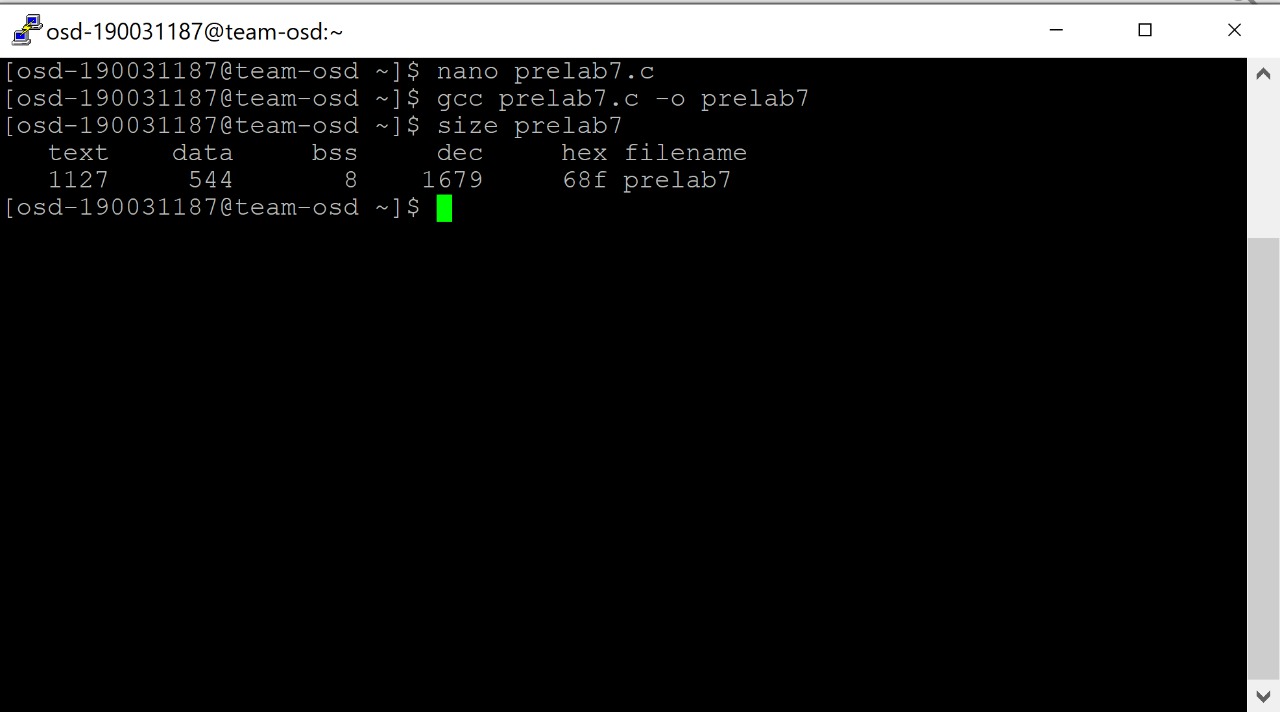
****

****

****

****

****

****

**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);**

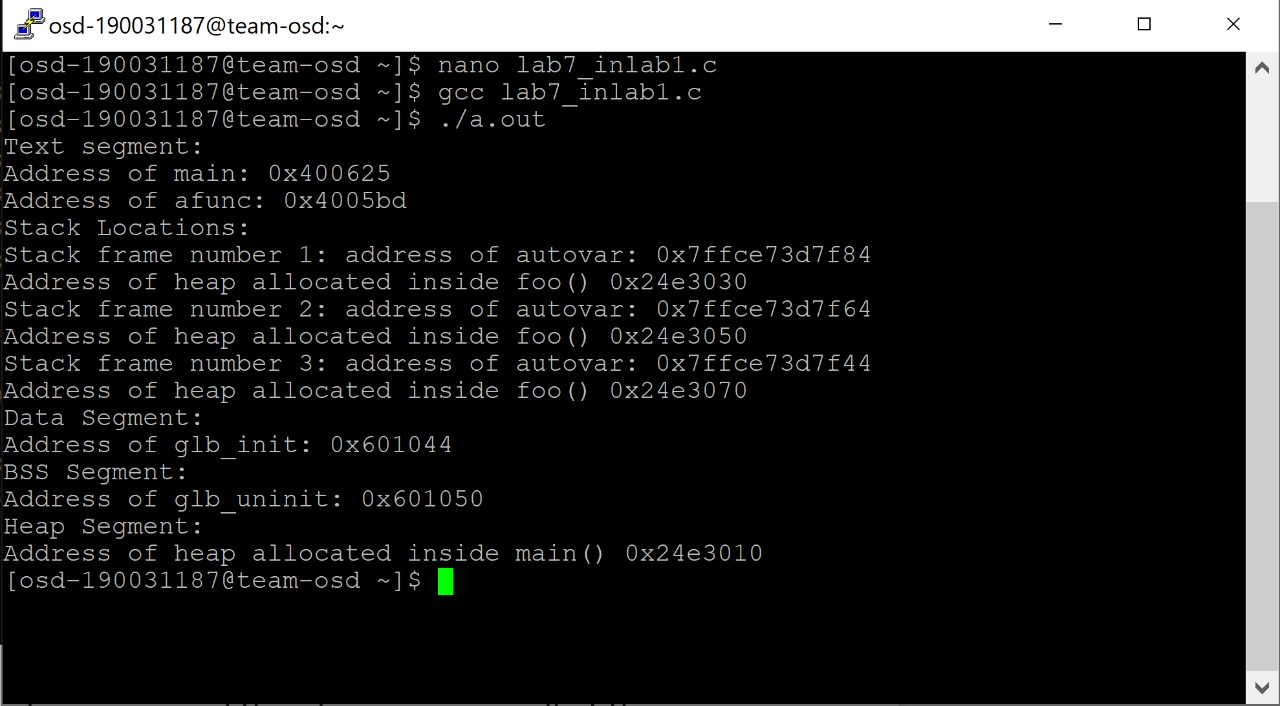
**printf("Heap Segment:\n");**

**printf("Address of heap allocated inside main() %p\n",ptr\_main);**

**return 0;**

**}**

**OUTPUT**



1. 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;**

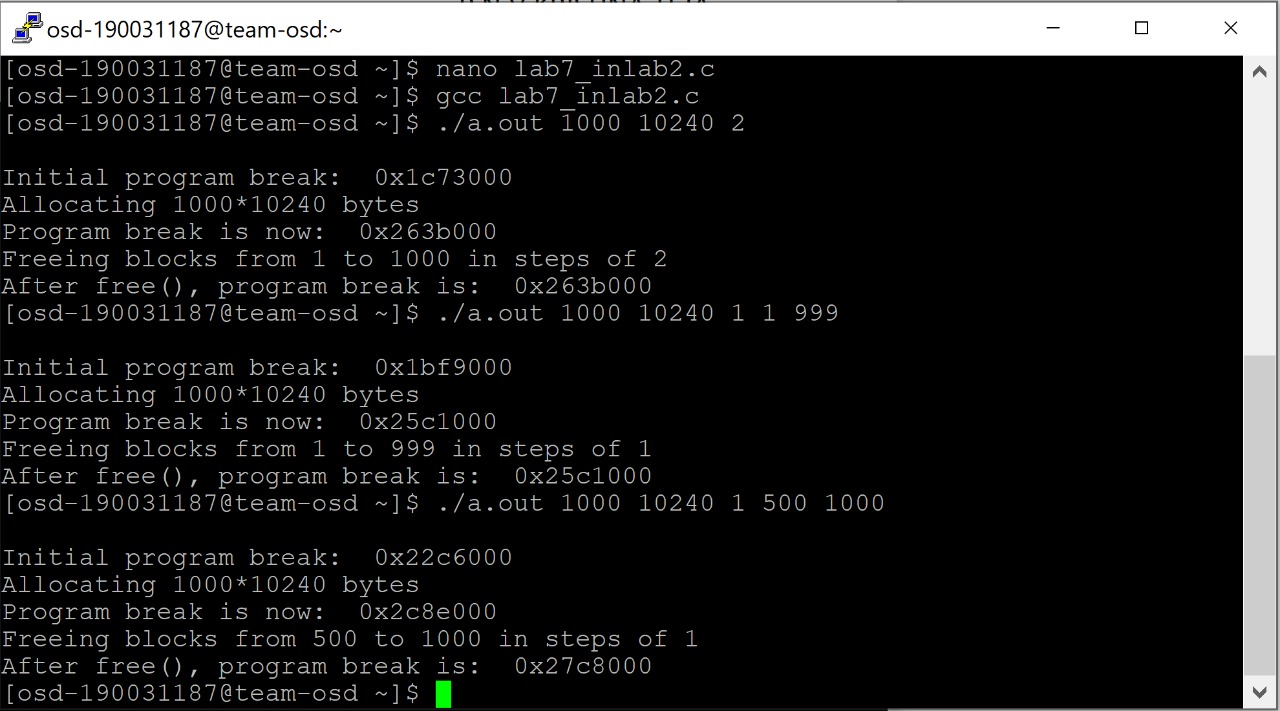
**j < freeMax;**

**j += freeStep)free(ptr[j]);**

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

**exit(10);**

**OUTPUT**



**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));**

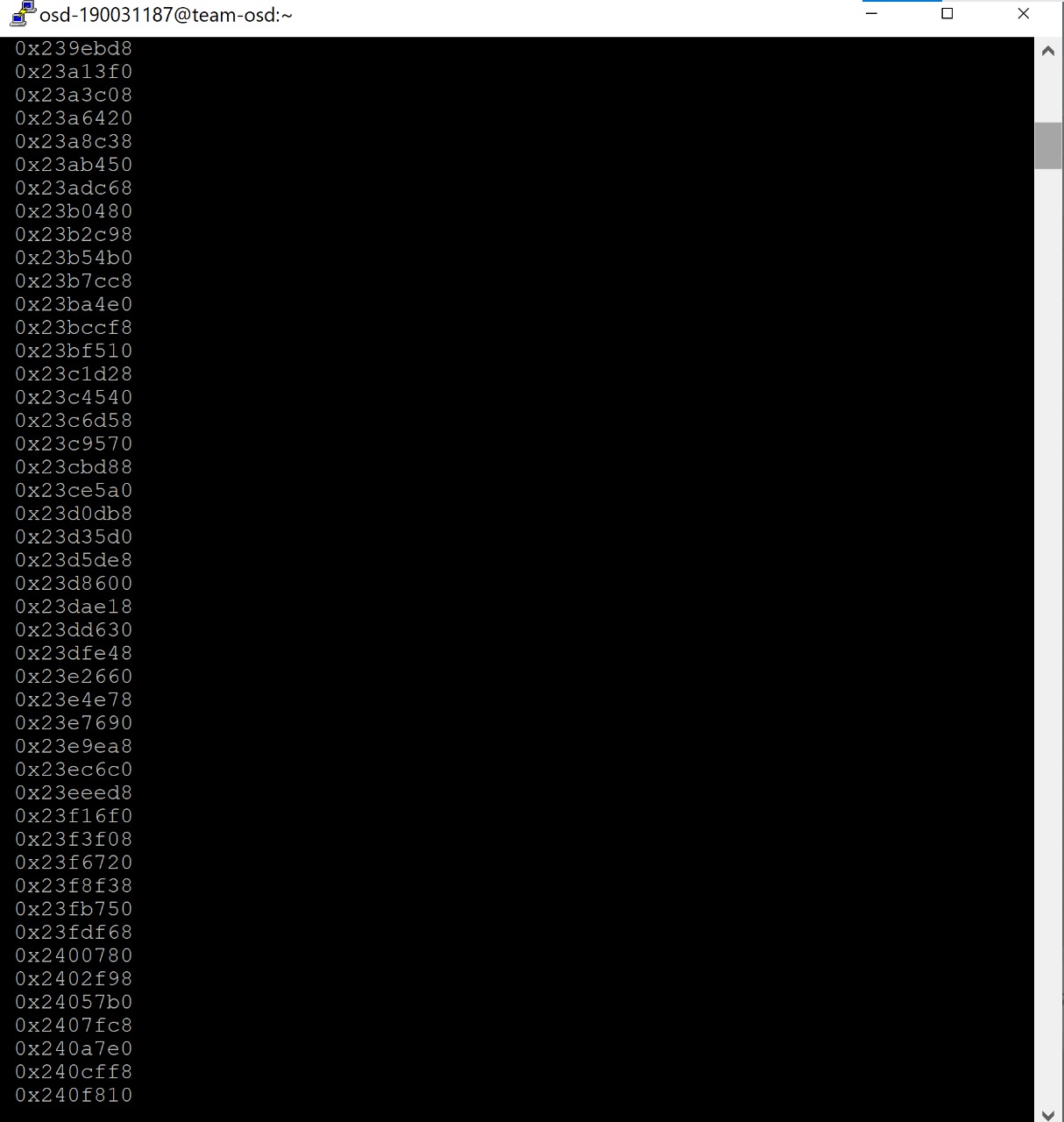
**printf("Freeing blocks from %d to %d in steps of %d\n",freeMin, freeMax, freeStep);**

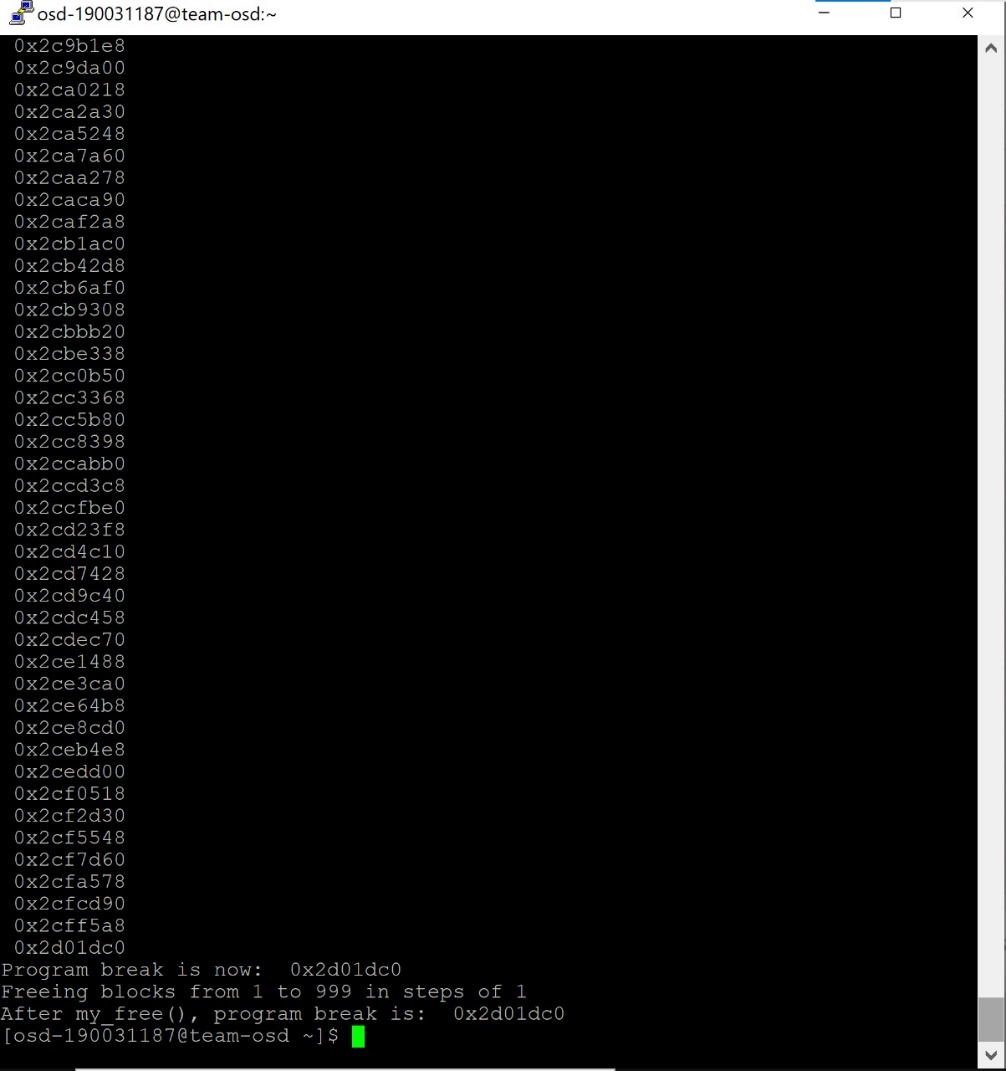
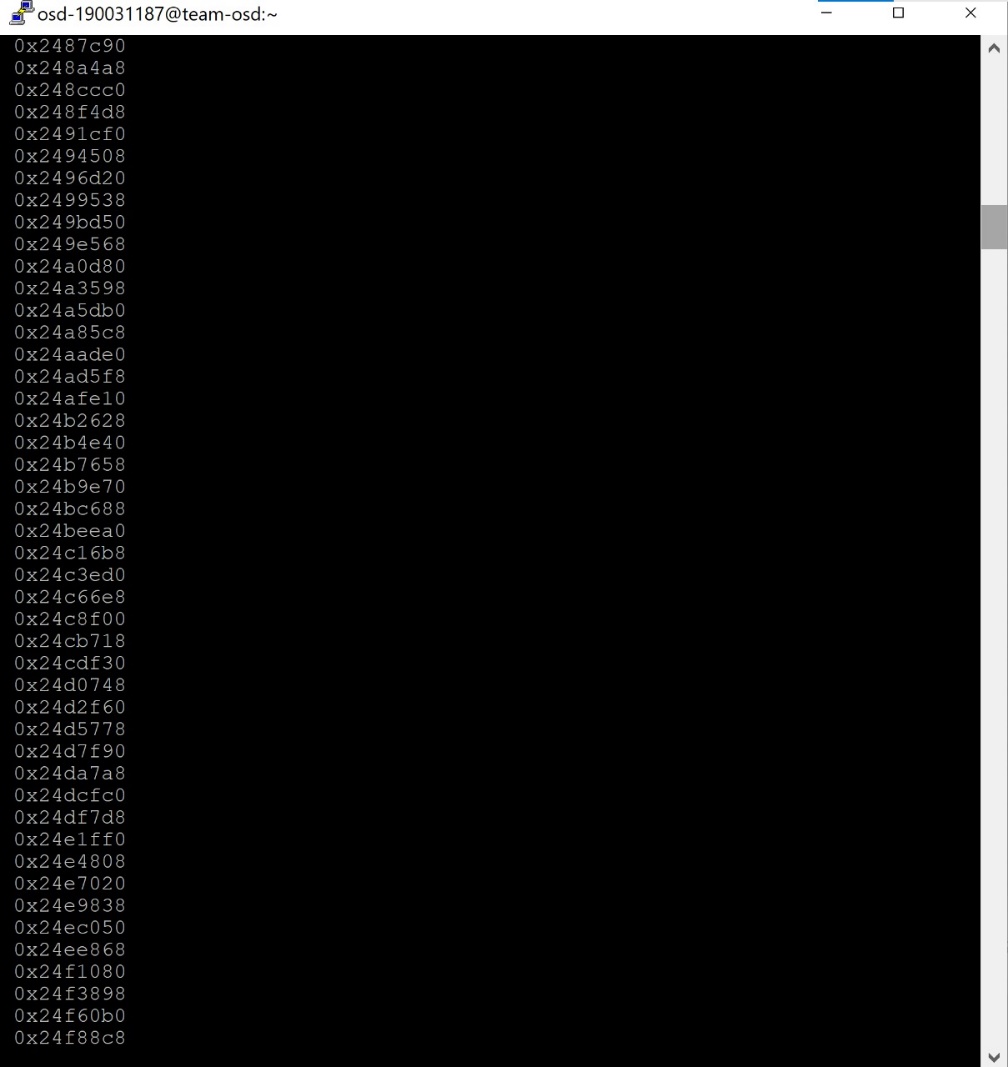
**for (j = freeMin -1; j < freeMax; j += freeStep) {my\_free(ptr[j]);}**

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

**exit(EXIT\_SUCCESS);}**

**OUTPUT**

****

****