/\*malloc.c

Implement ***malloc***  and ***free*** this is probably horrible c code.deficencies:\* it doesn't increase the page break by huge sizes to avoid too many system calls\* it doesn't lower the page break when memory gets freed\* it requires space to store pointers to the last and next block even for non-free blocksperformance\*/

#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);}

osdlab:~/programs$ ./a.out 1000 10240 1 1 999

Initial program break: 0x8ad0000

Allocating 1000\*10240 bytes0x8ad280c

0x8ad5018

0x8ad7824

0x8ada030

0x8adc83c

0x8adf048

0x8ae1854

0x8ae4060

0x8ae686c

0x8ae9078

0x8aeb884

0x8aee090

0x8af089c

0x8af30a8..

Program break is now: 0x9496ee0

Freeing blocks from 1 to 999 in steps of 1

After my\_free(), program break is: 0x9496ee0

