Q.1

#include <stdio.h>

#include <stdlib.h>

static void \* gBuffer;

static void \* gLoc;

const int gApplicationSize = 10000;

void \* (\*\_Alloc) (unsigned);

void \* OSOrCRTAllocator (int pApplicationSize)

{

return (void \*) malloc(pApplicationSize);

}

void \* CanAlloc (unsigned pSizeInBytes){

if (gApplicationSize - ((char \*)gLoc - (char \*)gBuffer) <= pSizeInBytes){

return NULL;

}

void \* ptr = gLoc;

gLoc = (char \*) gLoc + pSizeInBytes;

return ptr;

}

void \* CantAlloc (unsigned pSizeInBytes){

return NULL;

}

void Init (){

gBuffer = OSOrCRTAllocator (gApplicationSize);

gLoc = gBuffer;

if (gBuffer == NULL){

\_Alloc = &CantAlloc;

} else {

\_Alloc = &CanAlloc;

}

}

void \* Alloc (unsigned pSizeInBytes) {

if(gBuffer == NULL) {

Init();

}

return (\*\_Alloc)(pSizeInBytes);

}

void Free (void \* ptr) {

gLoc = ptr;

}

int main ()

{

int \* ptr1 = (int \*) Alloc (4);

char \* ptr2 = (char \*) Alloc (1);

double \* ptr3 = (double \*) Alloc (8);

/\*Free (ptr3);

Free (ptr2);

Free (ptr1);

free (gBuffer);

\*/return 0;

}

Q 1.2.

#include <stdio.h>

#include <stdlib.h>

static void \* gBuffer;

static void \* gLoc;

int gApplicationSize;

void \* (\*\_Alloc) (unsigned);

void \* OSOrCRTAllocator (int pApplicationSize)

{

return (void \*) malloc(pApplicationSize);

}

void \* CantAlloc (unsigned pSizeInBytes){

return NULL;

}

void Grow (unsigned pApplicaionSize) {

gApplicationSize += pApplicaionSize;

void \* temp\_buffer = OSOrCRTAllocator (gApplicationSize);

if(temp\_buffer == NULL) {

\_Alloc = &CantAlloc;

return;

}

for(int i = 0; i < ((char \*) gLoc - (char \*) gBuffer); ++i) {

char temp;

temp = \*((char \*)gBuffer) + i;

\*((char \*)gBuffer + i) = \*((char \*)temp\_buffer + i);

\*((char \*)temp\_buffer + i) = temp;

}

gLoc = (char \*)temp\_buffer + ((char \*)gLoc - (char \*)gBuffer);

free (gBuffer);

gBuffer = temp\_buffer;

}

void \* CanAlloc (unsigned pSizeInBytes){

if (gApplicationSize - ((char \*) gLoc - (char \*) gBuffer) <= pSizeInBytes){

Grow(gApplicationSize + pSizeInBytes);

return (\*\_Alloc)(pSizeInBytes);

}

void \* ptr = gLoc;

gLoc = (char \*) gLoc + pSizeInBytes;

return ptr;

}

void Init (unsigned pApplicationSize) {

gApplicationSize = pApplicationSize;

gBuffer = OSOrCRTAllocator (gApplicationSize);

gLoc = gBuffer;

if (gBuffer == NULL) {

\_Alloc = &CantAlloc;

} else {

\_Alloc = &CanAlloc;

}

}

void \* Alloc (unsigned pSizeInBytes) {

if(gBuffer == NULL) {

Init(10000);

}

return (\*\_Alloc)(pSizeInBytes);

}

void Free (void \* ptr) {

gLoc = ptr;

}

int main ()

{

int \* ptr1 = (int \*) Alloc (4);

char \* ptr2 = (char \*) Alloc (1);

double \* ptr3 = (double \*) Alloc (8);

char \* ptr4 = (char \*) Alloc (15000);

double \* ptr5 = (double \*) Alloc (8);

char \* ptr6 = (char \*) Alloc (15000);

printf("%p %p %p %p %p %p %p\n", ptr1, ptr2, ptr3, ptr4, ptr5, ptr6, gLoc);

Free (ptr3);

Free (ptr2);

Free (ptr1);

free (gBuffer);

return 0;

}

Q. 2

#include <stdio.h>

#include <stdlib.h>

static void \* gBuffer;

static void \* gLoc;

const int gApplicationSize = 10000;

void \* (\*\_Alloc) (unsigned);

void \* OSOrCRTAllocator (int pApplicationSize)

{

return (void \*) malloc(pApplicationSize);

}

void \* CanAlloc (unsigned pSizeInBytes){

if (gApplicationSize - ((char \*)gLoc - (char \*)gBuffer) <= pSizeInBytes){

return NULL;

}

void \* ptr = gLoc;

gLoc = (char \*) gLoc + pSizeInBytes;

return ptr;

}

void \* CantAlloc (unsigned pSizeInBytes){

return NULL;

}

void Init (){

gBuffer = OSOrCRTAllocator (gApplicationSize);

gLoc = gBuffer;

if (gBuffer == NULL){

\_Alloc = &CantAlloc;

} else {

\_Alloc = &CanAlloc;

}

}

void \* Alloc (unsigned pSizeInBytes) {

if(gBuffer == NULL) {

Init();

}

return (\*\_Alloc)(pSizeInBytes);

}

void Free (void \* ptr) {

gLoc = ptr;

}

int main ()

{

int \* ptr1 = (int \*) Alloc (4);

char \* ptr2 = (char \*) Alloc (1);

double \* ptr3 = (double \*) Alloc (8);

printf("%p %p %p %p\n", ptr1, ptr2, ptr3, gLoc);

Free (ptr3);

Free (ptr2);

Free (ptr1);

free (gBuffer);

return 0;

}