Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №1**

**з дисципліни «**[**Організація обчислювальних процесів**](http://wiki.kpi.ua/index.php/%D0%9E%D1%80%D0%B3%D0%B0%D0%BD%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F%20%D0%BE%D0%B1%D1%87%D0%B8%D1%81%D0%BB%D1%8E%D0%B2%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D1%85%20%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%96%D0%B2_(20402040))**»**

**на тему «Алокатор пам’яті загального призначення»**

Виконав:

студент ІІI курсу

групи ІО-53

Лісовий В. О.

Перевірив:

[ст.вик. Сімоненко А. В.](http://rozklad.kpi.ua/Schedules/ViewSchedule.aspx?v=7507d508-d0a2-4c01-a858-a063f8d724cc)

Київ – 2018

**Сирцевий код**

* **Allocator.h**

struct Header {

bool isFree;

size\_t nextSize;

size\_t prevSize;

};

void allocator\_init();

void \* mem\_alloc(size\_t);

void \* mem\_realloc(void \*, size\_t);

void mem\_free(void \*);

void allocator\_free();

void mem\_dump(const char[]);

* **Allocator.cpp**

#include "stdafx.h"

size\_t g\_size = 2000;

size\_t header\_size = sizeof(Header);

void \*memory;

Header \*first;

Header \*last;

Header \* nextHeader(Header \*header);

Header \* prevHeader(Header \*header);

Header \* getHeaderFromAddr(void \*address);

void allocator\_init() {

memory = malloc(g\_size);

first = (Header \*) memory;

first->isFree = false; // first free block

first->prevSize = 0;

first->nextSize = 0;

first = nextHeader(first);

first->isFree = true; // first free block

first->prevSize = 0;

first->nextSize = g\_size - 3 \* header\_size;

last = nextHeader(first); // last busy block

last->isFree = false;

last->prevSize = first->nextSize;

last->nextSize = 0;

}

void \* mem\_alloc(size\_t size) {

if (size <= 0)

return NULL;

Header\* header = first;

// search for first free block

while (!header->isFree && header != last && (header->nextSize - size) > header\_size) { // first free block that has more space that we need

header = nextHeader(header);

}

if (header == last) // no free space

return NULL;

size\_t buf = header->nextSize;

header->isFree = false; // mark block as busy

header->nextSize = size;

Header \*newHeader = nextHeader(header); // create new header

newHeader->isFree = true; // that is free

newHeader->prevSize = size;

newHeader->nextSize = buf - size - header\_size;

Header \*newNextHeader = nextHeader(newHeader);

if (newNextHeader->isFree) { // if next block after created is free - merge them

newHeader->nextSize += header\_size + newNextHeader->nextSize;

}

else { // else change prevSize

newNextHeader->prevSize = newHeader->nextSize;

}

return (void \*)((size\_t)header + header\_size);

}

void \* mem\_realloc(void \*addr, size\_t size) {

if (addr == NULL)

return mem\_alloc(size);

if (size <= 0)

return NULL;

Header \*current\_header = getHeaderFromAddr(addr);

int delta\_size = current\_header->nextSize - size;

if (delta\_size == 0) // no sense to move the block

return addr;

if (delta\_size > (int) header\_size) {

size\_t buf = current\_header -> nextSize;

current\_header->nextSize = size; // change block size

Header \*newHeader = nextHeader(current\_header); // create new header

newHeader->isFree = true; // that is free

newHeader->prevSize = size;

newHeader->nextSize = buf - size - header\_size;

nextHeader(newHeader)->prevSize = newHeader->nextSize;

return addr; // address doesn't change if block size is reduced

} else { // delete this block and call mem\_alloc. If it return Null, repair curr block

size\_t prevSizeBuf = current\_header->prevSize;

size\_t nextSizeBuf = current\_header->nextSize;

mem\_free(addr);

//mem\_dump("debug - free");

void \* result = mem\_alloc(size);

//mem\_dump("debug - malloc");

if (result != NULL) { // we have allocated memory

return result;

}

else { // if result is NULL - there is no free block

current\_header->isFree = false; // repair curr

current\_header->prevSize = prevSizeBuf;

current\_header->nextSize = nextSizeBuf;

prevHeader(current\_header)->nextSize = current\_header->prevSize; // repair prev block

nextHeader(current\_header)->prevSize = current\_header->nextSize; // repair next block

}

}

return NULL;

}

void mem\_free(void \*addr) {

Header \*current = getHeaderFromAddr(addr); // get \*Header of current block

Header \*next = nextHeader(current);

Header \*prev = prevHeader(current);

if (next->isFree && prev->isFree) { // megre 3 blocks

prev->nextSize += 2 \* header\_size + current->nextSize + next->nextSize;

nextHeader(prev)->prevSize = prev->nextSize;

} else if (next->isFree && !prev->isFree) { // merge next and curr

current->isFree = true;

current->nextSize += header\_size + next->nextSize;

nextHeader(current)->prevSize = current->nextSize;

} else if(!next->isFree && prev->isFree) { // merge prev and curr

prev->nextSize += header\_size + current->nextSize;

next->prevSize = prev->nextSize;

} else { // mark curr as free

current->isFree;

}

}

void allocator\_free() {

free(memory);

}

Header \* nextHeader(Header \*header) {

if (header == last) // only for test

return NULL;

return (Header \*) ((size\_t)header + header->nextSize + header\_size);

}

Header \* prevHeader(Header \*header) {

return (Header \*)((size\_t)header - header->prevSize - header\_size);;

}

Header \* getHeaderFromAddr(void \*address) {

return (Header \*)((size\_t)address - header\_size);

}

void mem\_dump(const char message[]) {

Header\* current = (Header \*) memory;

std::cout << "-----Out all Headers-----(" << message << ")\n";

while (current != NULL) {

std::cout << "isFree=" << (current->isFree ? "Free" : "Busy") << ", prevSize=" << current->prevSize

<< ", nextSize=" << current->nextSize << "\n";

current = nextHeader(current);

}

std::cout << "\n";

}

* **Os\_Lab1.cpp**

#include "stdafx.h"

using namespace std;

Header header1;

Header \*header;

void test();

int main()

{

allocator\_init();

test();

allocator\_free();

char end;

cin >> end;

return 0;

}

void test() {

mem\_dump("Start");

int \*arr = (int \*) mem\_alloc(100);

arr[0] = 0;

arr[1] = 1;

arr[2] = 2;

mem\_dump("malloc 100");

int \*arr1 = (int \*)mem\_alloc(50);

arr1[0] = 0;

arr1[1] = 1;

arr1[2] = 2;

mem\_dump("malloc 50");

arr = (int\*) mem\_realloc(arr, 38);

mem\_dump("realloc 100 to 38");

arr1 = (int\*)mem\_realloc(arr1, 200);

mem\_dump("realloc 50 to 200");

mem\_free(arr1);

mem\_dump("free 200");

mem\_free(arr);

mem\_dump("free 38");

}

**Результати тестування програми**

-----Out all Headers-----(Start)

isFree=Busy, prevSize=0, nextSize=0

isFree=Free, prevSize=0, nextSize=1964

isFree=Busy, prevSize=1964, nextSize=0

-----Out all Headers-----(malloc 100)

isFree=Busy, prevSize=0, nextSize=0

isFree=Busy, prevSize=0, nextSize=100

isFree=Free, prevSize=100, nextSize=1852

isFree=Busy, prevSize=1852, nextSize=0

-----Out all Headers-----(malloc 50)

isFree=Busy, prevSize=0, nextSize=0

isFree=Busy, prevSize=0, nextSize=100

isFree=Busy, prevSize=100, nextSize=50

isFree=Free, prevSize=50, nextSize=1790

isFree=Busy, prevSize=1790, nextSize=0

-----Out all Headers-----(realloc 100 to 38)

isFree=Busy, prevSize=0, nextSize=0

isFree=Busy, prevSize=0, nextSize=38

isFree=Free, prevSize=38, nextSize=50

isFree=Busy, prevSize=50, nextSize=50

isFree=Free, prevSize=50, nextSize=1790

isFree=Busy, prevSize=1790, nextSize=0

-----Out all Headers-----(realloc 50 to 200)

isFree=Busy, prevSize=0, nextSize=0

isFree=Busy, prevSize=0, nextSize=38

isFree=Busy, prevSize=38, nextSize=200

isFree=Free, prevSize=200, nextSize=1702

isFree=Busy, prevSize=1702, nextSize=0

-----Out all Headers-----(free 200)

isFree=Busy, prevSize=0, nextSize=0

isFree=Busy, prevSize=0, nextSize=38

isFree=Free, prevSize=38, nextSize=1914

isFree=Busy, prevSize=1914, nextSize=0

-----Out all Headers-----(free 38)

isFree=Busy, prevSize=0, nextSize=0

isFree=Free, prevSize=0, nextSize=1964

isFree=Busy, prevSize=1964, nextSize=0