НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Лабораторна робота №2

З дисципліни: «Об’єктно-орієнтоване програмування»

Виконав:

Студент групи КВ-51

Тимошенко Ігор

Перевірив

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КИЇВ 2016

mem\_dispatcher.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* file: mem\_dispatcher.h

\* author: Igor Tymoshenko

\* written: 27/11/2016

\* last modified: 27/11/2016

\* synopsis: memory dispatcher interface

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#define HEAP\_SIZE 10

typedef enum { FREE, ALLOCATED } STATUS;

typedef struct tag\_mem\_chunk {

int id;

int size;

STATUS status;

struct tag\_mem\_chunk\* next; //pointer to the next memory block

}mem\_chunk;

typedef struct {

int last\_id\_used;

mem\_chunk\* first; //pointer to the first memory block

}mem\_dispatcher;

//creates a heap as a single free block with id 0 and HEAP\_SIZE size

void init(mem\_dispatcher \*md);

//returns block id if allocated and -1 otherwise

int allocate(mem\_dispatcher \*md, int size);

//returns nonnegative value if block is deallocated and -1 otherwise

int deallocate(mem\_dispatcher \*md, int block\_id);

//reunites free blocks that were previously stored in various parts of a heap //into one successive block

void defragment(mem\_dispatcher \*md);

//displays heap status

void show\_memory\_map(mem\_dispatcher \*md);

mem\_dispatcher.c

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* file: mem\_dispatcher.c

\* author: Igor Tymoshenko

\* written: 27/11/2016

\* last modified: 27/11/2016

\* synopsis: realization of memory dispatcher interface

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "mem\_dispatcher.h"

//creates a heap as a single free block with id 0 and HEAP\_SIZE size

void init(mem\_dispatcher \*md) {

md->last\_id\_used = 0;

mem\_chunk \*tmp = (mem\_chunk \*)malloc(sizeof(mem\_chunk));

md->first = NULL;

tmp->id = 0;

tmp->size = HEAP\_SIZE;

tmp->status = FREE;

tmp->next = NULL;

md->first = tmp;

}

//returns block id if allocated and -1 otherwise

int allocate(mem\_dispatcher \*md, int size) {

int minID = -1;

int minSize;

if (size > HEAP\_SIZE || size <= 0)

return -1;

mem\_chunk \*tmp = md->first;

minSize = 10;

//searching in list a block with min free size

while (tmp) {

if ((tmp->status == FREE) && (tmp->size >= size) && (tmp->size <= minSize)) {

minSize = tmp->size;

minID = tmp->id;

}

tmp = tmp->next;

}

if (minID == -1)

return -1;

tmp = md->first;

while (minID != tmp->id) {

tmp = tmp->next;

}

tmp->size -= size;

tmp = (mem\_chunk \*)malloc(sizeof(mem\_chunk));

md->last\_id\_used++ ;

tmp->id = md->last\_id\_used;

tmp->size = size;

tmp->status = ALLOCATED;

tmp->next = md->first;

md->first = tmp;

return md->last\_id\_used;

}

//returns nonnegative value if block is deallocated and -1 otherwise

int deallocate(mem\_dispatcher \*md, int block\_id) {

mem\_chunk \*tmp, \*prev;

tmp = md->first;

prev = tmp;

// moving to block with id block\_id

while ( tmp && (tmp->id != block\_id) ) {

prev = tmp;

tmp = tmp->next;

}

if (tmp == NULL || tmp->status == FREE)

return -1;

tmp->status = FREE;

// union of adjacent free blocks

if ((tmp != prev) && (prev->status == FREE)) {

prev->size += tmp->size;

prev->next = tmp->next;

free(tmp);

tmp = prev;

}

prev = tmp;

tmp = tmp->next;

if ((tmp != NULL) && (tmp->status == FREE)) {

prev->size += tmp->size;

prev->next = tmp->next;

free(tmp);

}

return 0;

}

//reunites free blocks that were previously stored in various parts of a heap //into one successive block

void defragment(mem\_dispatcher \*md) {

int size\_free = 0;

mem\_chunk \*prev, \*tmp;

tmp = md->first;

// axception for first free blocks

while (tmp->status == FREE) {

md->first = tmp->next;

size\_free += tmp->size;

free(tmp);

tmp = md->first;

}

prev = tmp;

while (tmp->next != NULL) {

if (tmp->status == FREE) {

prev->next = tmp->next;

size\_free += tmp->size;

free(tmp);

tmp = prev;

}

prev = tmp;

tmp = tmp->next;

}

tmp->size += size\_free;

}

//displays heap status

void show\_memory\_map(mem\_dispatcher \*md) {

mem\_chunk \*tmp = md->first;

while (tmp) {

if ( tmp->status == FREE)

printf("\nblock id: %d\n size: %d status: %s\n", tmp->id, tmp->size, "free");

else

printf("\nblock id: %d\n size: %d status: %s\n", tmp->id, tmp->size, "allocate");

tmp = tmp->next;

}

}

test.c

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* file: test.c

\* author: Igor Tymoshenko

\* written: 27/11/2016

\* last modified: 27/11/2016

\* synopsis: memory dispatcher testing

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "mem\_dispatcher.h"

int main() {

int buf = 0;

mem\_dispatcher md;

init(&md);

do {

printf("\n1. Allocate\n2. Deallocate\n3. Show memory map\n4. Defragment\n0. Exit\nEnter command : ");

scanf("%d",&buf);

switch (buf) {

case 1:

printf("Enter block size: ");

scanf("%d", &buf);

if (allocate(&md, buf) == -1)

printf("Operation failed\n");

else

printf("Operation comlete\n");

break;

case 2:

printf("Enter block id: ");

scanf("%d", &buf);

if (deallocate(&md, buf) == -1)

printf("Operation failed\n");

else

printf("Operation complete\n");

break;

case 3:

show\_memory\_map(&md);

break;

case 4:

defragment(&md);

printf("Operation comlete\n");

break;

case 0:

return 0;

}

} while (1);

}