

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

Факультет прикладної математики

Кафедра системного програмування і спеціальних комп’ютерних систем

**Лабораторна робота №3**

з дисципліни “ Об’ектно-орієнтоване програмування (частина I) ”

Тема: «**Структури»**

**Варіант № 5**

Виконав:

студент 1 курсу ФПМ

групи КВ-41:

Горпинич-Радуженко

Іван Олександрович

**Перевірено:**

Київ 2015

***Постановка задачі***

Написати програму диспетчера пам'яті. Карта пам'яті представлена списком, кожний вузол якого містить опис блоку пам'яті - номер, розмір блоку, його статус (вільний, зайнятий) і покажчик на наступний блок.

block id: 0

size: 2 status: allocated

block id: 3

size: 2 status: free

block id: 2

size: 3 status: allocated

block id: 1

size: 3 status: free

У початковий момент у списку тільки один блок, що містить усю пам'ять.

block id: 0

size: 10 status: free

Виділення пам'яті виконується таким чином, що серед усіх вільних блоків, що мають розмір не менше необхідного, вибирається найменший. При звільненні блоку, суміжні блоки поєднуються. Наприклад, після звільнення блоку 2 одержимо

block id: 0

size: 2 status: allocated

block id: 3

size: 8 status: free

Передбачена операція дефрагментації пам'яті, при якій усі вільні блоки поєднуються в один.

block id: 0

size: 2 status: allocated

block id: 3

size: 2 status: free

block id: 2

size: 3 status: allocated

block id: 1

size: 3 status: free

Після дефрагментації одержимо

block id: 3

size: 5 status: free

block id: 0

size: 2 status: allocated

block id: 2

size: 3 status: allocated

***Текст програми***

**mem.h**

#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;

}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);

**mymem.c**

#include "mem.h"

int main()

{

int numcom,block\_id,size,flag=0;

mem\_dispatcher\* head = malloc(sizeof(mem\_chunk));

init(head);

while (flag == 0)

{

printf("1.Allocate\n");

printf("2.Deallocate\n");

printf("3.Show mem map\n");

printf("4.Defragment\n");

printf("5.Exit\n");

printf("Enter command: ");

scanf("%d", &numcom);

switch (numcom)

{

case 1:

printf("Enter size: ");

scanf("%d", &size);

head->last\_id\_used = allocate(head, size);

break;

case 2:

printf("Enter block\_id: ");

scanf("%d", &block\_id);

deallocate(head, block\_id);

break;

case 3:

show\_memory\_map(head);

break;

case 4:

defragment(head);

break;

case 5:

flag = -1;

break;

default:

printf("Wrong command\n");

}

printf("\n");

}

}

**mem.c**

#include "mem.h"

void init(mem\_dispatcher \*md)

{

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

temp->id = 0;

temp->size = HEAP\_SIZE;

temp->status = FREE;

temp->next = NULL;

md->last\_id\_used = 0;

md->first = temp;

}

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

{

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

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

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

int id\_use = md->last\_id\_used + 1, minsize = temp->size;

int flag = 0;

while (temp)

{

if (temp->status == FREE && temp->size >= size)

if (temp->size <= minsize)

{

buftemp = temp;

minsize = temp->size;

flag = 1;

}

if (temp->next == NULL) break;

temp = temp->next;

}

if (size == buftemp->size) {

buftemp->status = ALLOCATED;

return buftemp->id;

}

if (flag != 0)

{

buff->id = id\_use;

buff->size = size;

buff->status = ALLOCATED;

buff->next = NULL;

buftemp->size -= size;

temp->next = buff;

}

else printf("Allocation failed\n");

return(id\_use);

}

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

mem\_chunk \*pointer = md->first, \*p = NULL;

if (pointer->id == block\_id) {

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

pointer->size += pointer->next->size;

pointer->status = FREE;

pointer = pointer->next;

md->first->next = pointer->next;

free(pointer);

return 0;

}

pointer->status = FREE;

return 0;

}

while (pointer->next != NULL && pointer->next->id != block\_id)

pointer = pointer->next;

if (!pointer->next) return -1;

pointer->next->status = FREE;

if (pointer->status != FREE) pointer = pointer->next;

while (pointer->next != NULL && pointer->next->status == FREE) {

p = pointer->next;

pointer->size += p->size;

pointer->next = p->next;

free(p);

}

return 0;

}

void defragment(mem\_dispatcher \*md){

mem\_chunk \*pointer = md->first, \*point = NULL, \*p = NULL;

while (pointer != NULL && pointer->status != FREE)

pointer = pointer->next;

if (pointer == NULL) return;

point = pointer;

while (point->next != NULL) {

if (point->next->status == FREE) {

p = point->next;

pointer->size += p->size;

point->next = p->next;

free(p);

}

else point = point->next;

}

}

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

{

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

printf("\n");

while (temp)

{

printf("id : %d", temp->id);

printf(" size : %d", temp->size);

if (temp->status==FREE)

printf(" status : FREE" );

else printf(" status : allocated");

printf("\n");

temp=temp->next;

}

***Приклад роботи програми.***

1. Allocate

2. Deallocate

3. Show memory map

4. Defragment

5. Exit

Enter command: 3

block id: 0

size: 10 status: free

1. Allocate

2. Deallocate

3. Show memory map

4. Defragment

5. Exit

Enter command: 1

Enter block size: 15

Allocation failed

1. Allocate

2. Deallocate

3. Show memory map

4. Defragment

5. Exit

Enter command: 1

Enter block size: 3

1. Allocate

2. Deallocate

3. Show memory map

4. Defragment

5. Exit

Enter command: 1

Enter block size: 2

1. Allocate

2. Deallocate

3. Show memory map

4. Defragment

5. Exit

Enter command: 3

block id: 0

size: 5 status: free

block id: 2

size: 2 status: allocated

block id: 1

size: 3 status: allocated

1. Allocate

2. Deallocate

3. Show memory map

4. Defragment

5. Exit

Enter command: 1

Enter block size: 2

1. Allocate

2. Deallocate

3. Show memory map

4. Defragment

5. Exit

Enter command: 11

Wrong command

1. Allocate

2. Deallocate

3. Show memory map

4. Defragment

5. Exit

Enter command: 3

block id: 0

size: 3 status: free

block id: 3

size: 2 status: allocated

block id: 2

size: 2 status: allocated

block id: 1

size: 3 status: allocated