Міністерство освіти та науки України  
Національний технічний університет України  
«Київський політехнічний інститут»  
Факультет прикладної математики  
Кафедра системного програмування і спеціалізованих  
комп’ютерних систем

**Лабораторна робота №2**з дисципліни

**«Об’єктно орієнтоване програмування»**Тема: **«Структури»**

Виконав: Сахнік Іван Сергійович

Студент групи КВ-42

Перевірив(ла)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Київ-2015**

**Мета роботи** – опанувати способами проектування й використання структур, способами побудови й обробки списків на базі структур.

**3.2. Завдання на роботу.**

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. У початковий момент у списку тільки один блок, що містить усю пам'ять. 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. Передбачена операція дефрагментації пам'яті, при якій усі вільні блоки поєднуються в один.

***Текст програми:***

#include "mem\_dispatcher.h"

int main()

{

mem\_dispatcher \*md =(mem\_dispatcher\*)malloc(sizeof(mem\_dispatcher));

int size, block\_id, ch=0;

init(md);

while (ch != 5)

{

printf("1. Allocate :\n");

printf("2. Deallocate :\n");

printf("3. Show memory map :\n");

printf("4. Defragment :\n");

printf("5. Exit :\n");

printf("Enter command : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("\nEnter block size : ");

scanf("%d", &size);

if (allocate(md, size) == -1) printf("\nAllocation failed");

break;

case 2:

printf("\nEnter block id : ");

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

if (deallocate(md, block\_id) == -1) printf("\nDeallocated failed");

break;

case 3:

show\_memory\_map(md);

break;

case 4:

defragment(md);

break;

case 5: break;

default:

printf("\nWrong command");

}

printf("\n\n");

}

return 0;

}

#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);

#include "mem\_dispatcher.h"

void init(mem\_dispatcher \*md)

{

mem\_chunk \*pointer = NULL;

md->last\_id\_used = 0;

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

pointer->id = 0;

pointer->status = FREE;

pointer->next = NULL;

pointer->size = HEAP\_SIZE;

md->first = pointer;

}

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

{

mem\_chunk \*pointer = NULL, \*p\_t = NULL, \*new\_point = NULL;

int min = HEAP\_SIZE + 1;

pointer = md->first;

while (pointer)

{

if (pointer->size >= size && pointer->size < min && pointer->status == FREE)

{

min = pointer->size;

p\_t = pointer;

}

pointer = pointer->next;

}

if (!p\_t)

return -1;

if (size == p\_t->size)

{

p\_t->status = ALLOCATED;

return p\_t->id;

}

new\_point = (mem\_chunk\*)malloc(sizeof(mem\_chunk));

new\_point->next = p\_t->next;

new\_point->status = ALLOCATED;

p\_t->size = p\_t->size - size;

new\_point->size = size;

md->last\_id\_used++;

new\_point->id = md->last\_id\_used;

p\_t->next = new\_point;

return md->last\_id\_used;

}

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

{

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

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

{

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

{

pointer->size = 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 = pointer->size + p->size;

pointer->next = p->next;

free(p);

}

return 0;

}

void defragment(mem\_dispatcher \*md)

{

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

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

pointer = pointer->next;

p\_t = pointer;

while (p\_t->next != NULL)

{

if (p\_t->next->status == FREE)

{

p = p\_t->next;

pointer->size = pointer->size + p->size;

p\_t->next = p->next;

free(p);

}

else p\_t = p\_t->next;

}

}

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

{

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

while(start)

{

printf("block id: %d\n\t\tsize: ", start->id);

printf("%d \tstatus: ", start->size);

if (start->status == FREE)

printf("free\n");

else

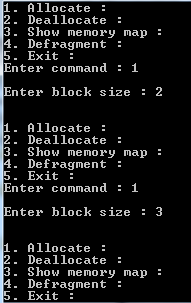
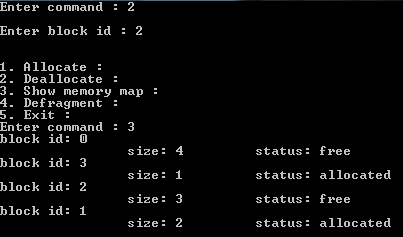
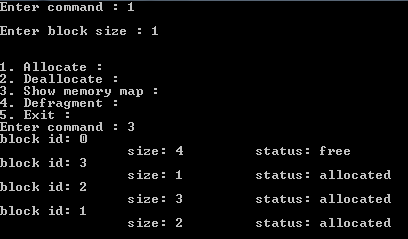
printf("allocated\n");

start = start->next;

}

}

***Тестування програми:***

******