**4 ЛАБОРАТОРНА РОБОТА №3**

**КЕРУВАННЯ ПАМ’ЯТЮ**

**4.1 Мета роботи**

Ознайомитися з основними алгоритмами виділення пам’яті, визначити переваги та недоліки кожного алгоритму.

**4.2 Виконання роботи**

#ifndef MEMORYMANAGER\_H

#define MEMORYMANAGER\_H

#include <QMutex>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

class MemoryManager

{

private:

typedef struct m\_block {

long size;

m\_block \*next;

bool isFree;

} MemoryBlock;

typedef MemoryBlock\* MemoryBlockPtr;

public:

MemoryManager(long size = 4096);

~MemoryManager();

void \*newMemory(long size);

void deleteMemory(void \*ptr);

private:

const int blockSize = sizeof(MemoryBlock);

MemoryBlockPtr firstBlock;

long lSize;

private:

void mergeFreeBlock();

MemoryBlockPtr findFreeBlock(long size);

void resizeAllMemmory(long size);

};

#endif // MEMORYMANAGER\_H

#include "Memorymanager.h"

MemoryManager::MemoryManager(long size)

{

this->lSize = size;

this->firstBlock = (MemoryBlockPtr)malloc(size + blockSize);

this->firstBlock->size = size;

this->firstBlock->next = nullptr;

this->firstBlock->isFree = true;

}

MemoryManager::~MemoryManager()

{

free(this->firstBlock);

}

void \*MemoryManager::newMemory(long size)

{

MemoryBlockPtr ptr = findFreeBlock(size);

if(ptr == nullptr) {

resizeAllMemmor(size);

ptr = findFreeBlock(size);

}

return (void\*)((long)ptr + blockSize);

}

void MemoryManager::deleteMemory(void \*ptr)

{

MemoryBlockPtr block = (MemoryBlockPtr)((long)ptr - blockSize);

block->isFree = true;

mergeFreeBlock();

}

void MemoryManager::mergeFreeBlock()

{

MemoryBlockPtr ptr = this->firstBlock;

while(ptr->next != nullptr) {

if(ptr->isFree && ptr->next->isFree) {

ptr->next = ptr->next->next;

ptr->size += ptr->next->size;

}

}

}

MemoryManager::MemoryBlockPtr MemoryManager::findFreeBlock(long size)

{

if(this->lSize < size) {

return nullptr;

}

MemoryBlockPtr current = this->firstBlock;

long findSize = size + blockSize;

while(current->next != nullptr) {

if(current->isFree && current->size >= findSize) {

MemoryBlockPtr newBlock = (MemoryBlockPtr)((long)current + blockSize +

current->size - findSize);

current->size -= findSize;

newBlock->size = size;

newBlock->isFree = false;

newBlock->next = current->next;

current->next = newBlock;

return newBlock;

}

}

return nullptr;

}

void MemoryManager::resizeAllMemor(long size)

{

MemoryBlockPtr block = (MemoryBlockPtr)malloc(this->lSize + size

+ blockSize);

memcpy(block, this->firstBlock, this->lSize);

this->firstBlock = block;

MemoryBlockPtr blkEnd = this->firstBlock;

while(blkEnd->next != nullptr) blkEnd = blkEnd->next;

lSize += size + blockSize;

blkEnd->size += size + blockSize;

}

**4.3 Висновки**

У ході виконання даної лабораторної роботи ознайомились з основними алгоритмами виділення пам’яті, визначили переваги та недоліки кожного алгоритму.