**МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»**

*Факультет прикладної математики*

*Кафедра «Спеціалізовані комп’ютерні системи»*

Лабораторна робота №1

з

СПЗ

**Виконав:** Шулик В.В. гр. КВ-81

м. Київ

2011

*Завдання для лабораторної роботи*

Разработать аллокатор общего назначения, используя за основу описанный выше базовый вариант алгоритма, принимая во внимания следующие условия:

1. Области памяти можно выделять любым доступным способом.
2. Функции mem\_alloc(), mem\_realloc() и mem\_free() должны соответствовать приведенным выше прототипам.
3. Адреса памяти, возвращаемые функциями mem\_alloc() и mem\_realloc(), должны быть выровнены на границу в 4 байта.
4. Попытаться уменьшить время поиска свободного блока памяти и время освобождения занятого блока.
5. Попытаться уменьшить фрагментацию памяти.
6. Написать функцию mem\_dump(), которая должна выводить на консоль состояние областей памяти.
7. **Опис алгоритму:**

Під час ініціалізації аллокатору виділяється фіксовано область пам’яті (за замовчуванням 2048 байт), вказівник на неї записується у спеціально створений вектор. Ця область пам’яті реалізована у вигляді структури даних, що складається з масиву необхідного розміру та змінної що зберігає кількість блоків, розміщених в цьому масиві. Автоматично в цьому масиві створюється блок пам’яті,що займає весь виділений об’єм, з міткою «вільно».

Функція №№№ створює у масиві блок необхідного розміру(відтинаючи його від існуючого вільного блоку),додатково записуючи перед цим блоком 8 байтів службової інформації,такої як індикатор зайнятості поточного блоку,розмір поточного блоку,розміри попереднього і наступного блоків(якщо вони існують). У випадку заповнення виділеної області пам’яті (або недостатньої кількості вільної в ній пам’яті) створюється нова, а вказівник на неї записується у вектор. Знищення блоку пам’яті відбувається шляхом запису до відповідних байтів ознаки звільнення блоку, або, якщо звільнена пам'ять зливається з сусідніми вільними блоками,додаткова інформація записується до них. При зміні розміру блоку пам’яті визначається, чи є попереду вільний блок пам’яті,який має бажаний розмір, якщо так – значить блок розширяється на тому ж місці за рахунок сусіднього вільно,якщо - ні, то пам’яті під нього виділяється на новому місці,а старий блок знищується.

1. **Оцінка алгоритму:**

Для кожного виділеного блоку пам’яті створюється додатково 8 байт інформації,такої як ознака зайнятості блоку,розмір поточного блоку,розмір попереднього і наступного блоків. Отже для Н блоків необхідно Н\*8 байт додаткової пам’яті.

Для пошуку необхідного блоку необхідно перевірити К заголовків,де К – це кількість об’єктів необхідної області пам’яті. Для знаходження потрібної області пам’яті в векторі необхідно виконати максимум М ітерацій, де М – це кількість елементів вектора( областей пам’яті).

1. **Недоліки і переваги алгоритму.**

Цей алгоритм не оптимально використовувати для виділення маленьких об’ємів пам’яті так як залучається багато місця для збереження додаткової інформації. В свою чергу для виділення великих обсягів(блоків) пам’яті він працює досить швидко,адже використовує для сховища масив,який забезпечує швидкий доступ до будь-якої комірки(на відміну ,наприклад, від списків, які б було доцільно використовувати в алгоритмі для алокації малих об’ємів пам’яті).

1. **Текст програми:**

--------main.cpp---------

#include "myHeader.h"

int main()

{

void\* s1;

void\* s2;

void\* s3;

void\* s4;

void\* s5,\*s6, \*s7, \*s8, \*s9;

short i = DEFAULT\_PORTION\_SIZE;

memAlloc\* ma = new memAlloc();

ma->nullFile();

s1 = ma->mem\_alloc(100);

s2 = ma->mem\_alloc(100);

s3 = ma->mem\_alloc(100);

s4 = ma->mem\_alloc(100);

ma->mem\_dump();

ma->mem\_free(s1);

ma->mem\_realloc(s2,400);

ma->mem\_dump();

s9 = ma->mem\_alloc(1000);

ma->mem\_dump();

s5 = ma->mem\_alloc(1000);

ma->mem\_dump();

s6 = ma->mem\_alloc(200);

s7 = ma->mem\_alloc(200);

s8 = ma->mem\_alloc(100);

ma->mem\_dump();

ma->mem\_free(s6);

ma->mem\_free(s3);

ma->mem\_free(s4);

ma->mem\_dump();

return 0 ;

}

--------myCode.cpp------

#include "myHeader.h"

void memPortion::init()

{

pData = new unsigned short[DEFAULT\_PORTION\_SIZE];

biggestPart = (unsigned short)(DEFAULT\_PORTION\_SIZE - 4);

\*(pData + 0) = (unsigned short)IS\_AVAILABLE;

\*(pData + 1) = (unsigned short)biggestPart;

\*(pData + 2) = (unsigned short)0;

\*(pData + 3) = (unsigned short)0;

numBlocks = 1 ;

}

void\* memPortion::allocate(std::size\_t size\_f)

{

unsigned short size = size\_f;

//vuravnyuvannya

if( size % 4 == 1)

size += 3;

if (size % 4 == 2)

size += 2;

if (size % 4 == 3)

size += 1;

size = size/2;

//end vuravnyuvannya

if (size > biggestPart)

return NULL;

unsigned short\* p = pData;

//while ( \*(p+3) != 0 )

for (int i = 0; i < numBlocks ; i++)

{

// std::cout <<"\*\*inCicle\*\*";

if ( \*p == (unsigned short)IS\_AVAILABLE )

//if ( \*p == 1 )

{

//std::cout <<"\*\*OP\*\*";

if ( \*(p+1) == size )

{

\*p = IS\_NOT\_AVAILABLE;

//numBlocks++;

//std::cout <<"return1";

return p;

}

if ( (\*(p+1) > size) && (\*(p+1) < (size + 6)))

{

\*p = IS\_NOT\_AVAILABLE;

//numBlocks++;

//std::cout <<"return2";

return p;

}

if (\*(p+1) >= (size + 6))

{

//nastupnuy block

if ( \*(p+3) != 0 )

{

\*(p+4+\*(p+1)+2) = \*(p+1) - size - 4;

}

//vilna chastuna

\*(p + 4 + size + 0) = (unsigned short)IS\_AVAILABLE;

\*(p + 4 + size + 1) = \*(p+1) - size - 4;

\*(p + 4 + size + 2) = size;

\*(p + 4 + size + 3) = \*(p+3);

//te sho zanyalu

\*(p + 0) = (unsigned short)IS\_NOT\_AVAILABLE;

\*(p + 1) = size;

//\*(p + 2) =

\*(p + 3) = \*(p + 4 + size + 1);

//poperedniy block

if ( \*(p+2) != 0 )

{

\*(p-\*(p+2) - 1 ) = \*(p+1);

}

numBlocks++;

return (p);

}

}

//std::cout << "next";

p = p + 4 + \*(p+1);

}

return NULL;

}

void memPortion::deAllocate(void\* pp)

{

unsigned short\* p = (unsigned short\*)pp;

\*(p + 0) = (unsigned short)IS\_AVAILABLE;

cout<<numBlocks<<"\n";

if ( (\*(p+3) != 0) && ((\*(p + 4 + \*(p+1))) == IS\_AVAILABLE ))

{

std::cout << "jj1\n";

unsigned short buf = \*(p+1);

\*(p+1) = \*(p+1) + 4 + \*(p+3);

\*(p+3) = \*(p + 4 + buf + 3);

if ( \*(p+2) != 0 )

\*(p - \*(p+2) - 1) = \*(p+1);

if( \*(p + 3) != 0)

\*(p + 4 + \*(p+1) + 2) = \*(p+1);

//numBlocks--;

numBlocks--;

cout<<numBlocks<<"\n";

//numBlocks--;

};

if ( (\*(p+2) != 0) && (\*((p - \*(p+2)-4)) == IS\_AVAILABLE) )

{

std::cout << "jj2\n";

\*(p - \*(p+2)-3) = \*(p+1) +\*(p - \*(p+2)-3) +4;

\*(p - \*(p+2)-1) = \*(p+3);

if ( \*(p - \*(p+2) -2) != 0)

\*(p - \*(p+2) - 4 - \*(p - \*(p+2)-2) - 1) = \*(p - \*(p+2) - 3);

//if ( \*(p - \*(p+2) - 1) != 0)

if ( \*(p+3) != 0)

\*(p + 4 + \*(p+1) + 2) = \*(p - \*(p+2) - 3);

numBlocks--;

}

pp = NULL;

//cout<<"####"<<pp;

}

void\* memPortion::reallocate(void\* pp, size\_t size\_f)

{

bug = 0;

unsigned short\* del = pData;

for ( int i = 0; i < numBlocks; i++ )

{

if ( ( pp == del ) && ( \*del == IS\_AVAILABLE ) )

{

cout<<"Fuck";

bug = 1;

return NULL;

}

del = del + \*(del+1) + 4;

}

unsigned short size = size\_f;

//vuravnyuvannya

if( size % 4 == 1)

size += 3;

if (size % 4 == 2)

size += 2;

if (size % 4 == 3)

size += 1;

size = size/2;

unsigned short\* p = (unsigned short\*)pp;

if ( size <= \*(p+1))

{

if ( (\*(p+1) - size) > 6 )

{

if ( \*(p+2) != 0 )

\*( p - \*(p+2)-1) = size;

if ( (\*(p+3) != 0) && (\*(p +4 + \*(p+1)) == IS\_NOT\_AVAILABLE) )

{

\*( p + 4 +\*(p+1)+ 2) = \*(p+1) - size-4;

\*( p + 4 + size + 0) = IS\_AVAILABLE;

\*( p + 4 + size + 1) = \*(p+1) - size - 4;

\*( p + 4 + size + 2) = size;

\*( p + 4 + size + 3) = \*(p+3);

numBlocks++;

}

if ( (\*(p+3) != 0) && (\*(p +4 + \*(p+1)) == IS\_AVAILABLE) )

{

\*( p + 4 + size + 0 ) = IS\_AVAILABLE;

\*( p + 4 + size + 1 ) = \*(p+1) - size - 4 + \*(p+3) + 4;

\*( p + 4 + size + 2 ) = size;

//\*( p + 4 + size + 3 ) = \*(p + 4 + \*(p+1) +4 + \*(p+3) +1);

\*( p + 4 + size + 3 ) = \*(p + 4 + \*(p+1) +3);

if ( \*(p + 4 + \*(p+1) +3) != 0 )

\*(p +4 + \*(p+1) + 4 + \*(p+3) +2) = \*( p + 4 + size + 1 );

}

if ( \*(p+3) == 0 )

{

\*( p + 4 + size + 0 ) = IS\_AVAILABLE;

\*( p + 4 + size + 1 ) = \*(p+1) - size - 4;

\*( p + 4 + size + 2 ) = size;

\*( p + 4 + size + 3 ) = 0;

numBlocks++;

}

\*(p+3) = \*( p + 4 + size + 1 );

\*(p+1) = size;

return p;

}

return p;

}

if ( (size > \*(p+1)) && (\*(p+3) != 0) )

{

if ( \*(p+4+\*(p+1)) == IS\_AVAILABLE && ( size <= (\*(p+1) + \*(p+3))) )

{

if (( \*(p+1) + \*(p+3) - size +4) >= 6)

{

if ( \*(p+2) != 0 )

\*(p - \*(p+2) - 1) = size;

unsigned short dd = \*(p+4+\*(p+1)+3);

\*(p+3) = (\*(p+1) + \*(p+3) - size );

\*(p+1) = size;

//\*(p+2)

\*( p + 4 + size + 0 ) = IS\_AVAILABLE;

\*( p + 4 + size + 1 ) = \*(p+3);

\*( p + 4 + size + 2 ) = size;

\*( p + 4 + size + 3 ) = dd;

if ( \*( p + 4 + size + 3 ) != 0)

\*(p+4+size+4+\*(p+3)+2) = \*(p+3);

//numBlocks++;

return p;

}

if ( \*(p+2) != 0 )

\*(p - \*(p+2) - 1) = \*(p+1) + \*(p+3) +4;

unsigned short dd = \*(p+1);

\*(p+1) = \*(p+1) + \*(p+3) +4;

//\*(p+2)=

\*(p+3) = \*(p+4+dd+3);

if ( \*(p+3) != 0)

\*(p + 4 + \*(p+1) +2) = \*(p+1);

numBlocks--;

return p;

}

}

cout<<"fuck";

return NULL;

}

void memPortion::picture()

{

unsigned short\* p = pData;

for (int i = 0; i < numBlocks; i ++)

{

printf("|%d|%3.1d|%3.1d|%3.1d|",\*p,\*(p+1),\*(p+2),\*(p+3));

if (\*(p) == 1 )

std::cout << "\_";

else

std::cout << "\*";

p = p + 4 + \*(p+1);

}

std::cout << "\n";}

void memPortion::pictureFile()

{

unsigned short\* p = pData;

ofstream f("c:\\myfile.dat", ios::app);

//ofstream myFile;

if ( f == false)

{

cerr<<"File can't be open"<<endl;

exit(1);

}

for ( int i = 0; i < numBlocks; i++ )

{

//myFile << "|"<<\*(p)<<"|"<<\*(p+1)<<"|"<<\*(p+2)<<"|"<<\*(p+3)<<"|";

f<<"|";

f<<\*(p);

f<<"|";

f.width(4);

f<<\*(p+1);

f<<"|";

f.width(4);

f<<\*(p+2);

f<<"|";

f.width(4);

f<<\*(p+3);

f<<"|";

if (\*(p) == 1 )

f << "free|";

else

f << "keep|";

p = p + 4 + \*(p+1);

}

f<< "%endOfPart%\n";

}

void memPortion::nullFile()

{

ofstream myFile("c:\\myfile.dat", ios::out);

myFile<<"";

}

void memPortion::nextLineFile()

{

ofstream myFile("c:\\myfile.dat", ios::app);

myFile<<"goToNextOutput----------🡪\n";

}

void\* memAlloc::mem\_alloc(std::size\_t size)

{

memPortion\* mp = NULL;

void\* np;

if ( portions.size() == 0)

{

mp = new memPortion();

mp->init();

portions.push\_back(mp);

}

for ( int i = 0; i < portions.size(); i++)

{

mp = portions[i];

np = mp->allocate(size);

if ( np != NULL)

{

return np;

}

}

mp = new memPortion();

mp->init();

portions.push\_back(mp);

np = mp->allocate(size);

return np;

}

void memAlloc::mem\_free(void \*p)

{

//unsigned short\* pp = (unsigned short\*)p;

memPortion\* mp = NULL;

for (int i = 0; i < portions.size(); i++ )

{

mp = portions[i];

if ( (p >= mp->pData) && ( p < ( (mp->pData) + DEFAULT\_PORTION\_SIZE ) ) )

{

cout<<i<<"qqqqq";

mp->deAllocate(p);

return;

}

}

return;

}

void\* memAlloc::mem\_realloc(void \*p, std::size\_t size)

{

memPortion\* mp = NULL;

void\* pp;

for (int i = 0; i < portions.size(); i++ )

{

mp = portions[i];

if ( (p >= mp->pData) && ( p < ( mp->pData + DEFAULT\_PORTION\_SIZE ) ) )

{

pp = mp->reallocate(p, size);

if ( mp->bug == 1)

return NULL;

if ( pp != NULL)

{

return pp;

}

}

}

pp = mem\_alloc(size);

mem\_free(p);

return pp;

}

void memAlloc::mem\_dump()

{

memPortion\* mp;

//nullFile();

int i;

for (i = 0; i < portions.size(); i++)

{

mp = portions[i];

mp->pictureFile();

}

if (i == 0)

return;

mp->nextLineFile();

}

void memAlloc::nullFile()

{

ofstream myFile("c:\\myfile.dat", ios::out);

myFile<<"";

}

--------myHeader.h-------

#include <vector>

#include <iostream>

#include <fstream>

#include <stdlib.h>

using namespace std;

#ifndef DEFAULT\_PORTION\_SIZE

#define DEFAULT\_PORTION\_SIZE 1024

#endif

#ifndef IS\_AVAILABLE

#define IS\_AVAILABLE 1

#endif

#ifndef IS\_NOT\_AVAILABLE

#define IS\_NOT\_AVAILABLE 0

#endif

struct memPortion

{

void init();

void picture();

void\* allocate(std::size\_t size);

void deAllocate(void\* p);

void\* reallocate(void \*p, size\_t size);

void pictureFile();

void nullFile();

void nextLineFile();

unsigned short\* pData;

//the biggest block in the portion:

unsigned short biggestPart;

unsigned short numBlocks;

int bug;

};

class memAlloc

{

private:

//typedef std::vector<memPortion> Portions;

//Portions\* portions;

vector<memPortion\*> portions;

public:

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

void mem\_free (void\* p);

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

void mem\_dump();

void nullFile();

};

*Результат роботи програми :*

|0| 50| 0| 50|keep||0| 50| 50| 50| keep ||0| 50| 50| 50| keep ||0| 50| 50| 804| keep ||1| 804| 50| 0| free |%endOfPart%

goToNextOutput----------🡪

|1| 104| 0| 50| free ||0| 50| 104| 50| keep ||0| 50| 50| 200| keep ||0| 200| 50| 600| keep ||1| 600| 200| 0| free |%endOfPart%

goToNextOutput----------🡪

|1| 104| 0| 50| keep ||0| 50| 104| 50| keep ||0| 50| 50| 200| keep ||0| 200| 50| 500| keep ||0| 500| 200| 96| keep ||1| 96| 500| 0| free |%endOfPart%

goToNextOutput----------🡪

|1| 104| 0| 50|FREE||0| 50| 104| 50| keep ||0| 50| 50| 200| keep ||0| 200| 50| 500| keep ||0| 500| 200| 96| keep ||1| 96| 500| 0| free |\* END OF PART \*

|0| 500| 0| 516| keep ||1| 516| 500| 0| free |%endOfPart%

goToNextOutput----------🡪

|0| 104| 0| 50| keep ||0| 50| 104| 50| keep ||0| 50| 50| 200| keep ||0| 200| 50| 500| keep ||0| 500| 200| 50| keep ||0| 50| 500| 42| keep ||1| 42| 50| 0| free |%endOfPart%

|0| 500| 0| 100| keep ||0| 100| 500| 412| keep ||1| 412| 100| 0| free |%endOfPart%

goToNextOutput----------🡪

|1| 212| 0| 200| free ||0| 200| 212| 500| keep ||0| 500| 200| 50| keep ||0| 50| 500| 42| keep ||1| 42| 50| 0| free |%endOfPart%

|0| 500| 0| 100| keep ||0| 100| 500| 412| keep ||1| 412| 100| 0|free|%endOfPart%

goToNextOutput----------🡪