Національний технічний університет України

«Київський політехнічний інститут»

**Факультет інформатики та обчислювальної техніки**

**Кафедра обчислювальної техніки**

**Лабораторна робота №1**

студента групи ІО-93

Данильчук В. О.

**2012**

**public** **class** mem {

**public** mem(**int** memSize) {

**if** (memSize < 10)

memSize = 10;

memory = **new** **int**[memSize];

memory[0] = 1;

memory[1] = 3;

memory[2] = 0;

memory[4] = memSize - 6;

memory[5] = 3;

memory[memSize - 3] = 1;

memory[memSize - 2] = 0;

memory[memSize - 1] = memSize - 3;

}

//выдиление памяти

**public** **int** mem\_alloc(**int** size) {

**if** (size < 1)

size = 1;

**int** index = getNextPtr(0);

**int** ptr = -1;

**while** (getOffsetN(index) != 0) {

**if** ((getHeader(index) != 1) && ((getOffsetN(index) - 3) >= size)) {

setHeader(index, 1);

**if** ((getOffsetN(index) - 7) >= size) {

**int** Nptr = index + size + 3;

**int** next = getNextPtr(index);

**int** dCs = next - Nptr;

**int** dPs = Nptr - index;

setHeader(Nptr, 0);

setOffsetP(next, dCs);

setOffsetN(Nptr, dCs);

setOffsetN(index, dPs);

setOffsetP(Nptr, dPs);

}

**return** index;

}

index = getNextPtr(index);

}

**return** ptr;

}

//Освобождение памяти

**public** **void** mem\_free(**int** ptr) {

**int** ps = getPrevPtr(ptr);

**if** (getHeader(ps) != 1) {

ps = getPrevPtr(ps);

**if** (getHeader(ps) != 1)

System.*err*.println("ERROR offset prev ");

}

**int** cs = getNextPtr(ptr);

**if** (getHeader(cs) != 1) {

cs = getNextPtr(cs);

**if** (getHeader(cs) != 1)

System.*err*.println("ERROR offset next ");

}

**int** freePtr = getNextPtr(ps);

setHeader(freePtr, 0);

**int** dCs = cs - freePtr;

setOffsetN(freePtr, dCs);

setOffsetP(cs, dCs);

setHeader(ptr, 0);

**for** (**int** i = 3; i < getOffsetN(freePtr); i++)

memory[freePtr + i] = 0;

}

//изменение розмеров выделеной памяти

**public** **int** mem\_realloc(**int** ptr, **int** size) {

**if** (size == 0)

size = 1;

**int** csize = getOffsetN(ptr) - 3;

**int** d = Math.*abs*(csize - size);

**if** (csize == size)

**return** ptr;

**if** (csize > size) { //уменшение

**int** index = ptr + size + 3;

**int** next = getNextPtr(ptr);

**if** (getHeader(next) != 1) {//с корекцией пустой ячейки с права

**int** oldCs = getOffsetN(next);

**int** oldPs = getOffsetP(next);

setHeader(index, 0);

setOffsetN(index, oldCs + d);

setOffsetP(index, oldPs - d);

setOffsetN(ptr, getOffsetN(ptr) - d);

next = getNextPtr(index);

setOffsetP(next, getOffsetP(next) + d);

**for** (**int** i = 3; i < getOffsetN(index); i++)

memory[index + i] = 0;

} **else** {

**if** (d > 3) {//с созданием пустой ячейки с права

setHeader(index, 0);

setOffsetN(index, next - index);

setOffsetP(index, size + 3);

setOffsetP(next, next - index);

setOffsetN(ptr, size + 3);

**for** (**int** i = 3; i < getOffsetN(index); i++)

memory[index + i] = 0;

}

}

**return** ptr;

} **else** {//увеличение

**int** next = getNextPtr(ptr);

**if** (getHeader(next) != 1) {//если с права пусто

**if** (getOffsetN(next) >= d) {//если места достаточно

**if** ((getOffsetN(next) - d) > 3) {//с созданием пустой ячейки

**int** oldCs = getOffsetN(next);

**int** oldPs = getOffsetP(next);

next += d;

setOffsetN(next, oldCs - d);

setOffsetP(next, oldPs + d);

setHeader(next, 0);

setOffsetN(ptr, getOffsetN(ptr) + d);

next = getNextPtr(next);

setOffsetP(next, getOffsetP(next) - d);

**for** (**int** i = getOffsetN(ptr) - d; i < getOffsetN(ptr); i++)

memory[ptr + i] = 0;

} **else** {

**int** oldCs = getOffsetN(ptr);

**int** cs = getOffsetN(ptr) + getOffsetN(next);

setOffsetN(ptr, cs);

next = getNextPtr(ptr);

setOffsetP(next, cs);

**for** (**int** i = oldCs; i < getOffsetN(ptr); i++)

memory[ptr + i] = 0;

}

} **else** {

**int** prev = getPrevPtr(ptr);

**if** (getHeader(prev) != 1) {//если с лева пусто

**if** (getOffsetN(next) + getOffsetN(prev) >= d) {//если места с лева и права достаточно

**if** ((getOffsetN(next) + getOffsetN(prev) - d) > 3) {//с созданием

**int** oldCs = getOffsetN(ptr);

**int** delta = d - getOffsetN(next);

setOffsetN(prev, getOffsetN(prev) - delta);

setOffsetP(ptr, getOffsetP(ptr) - delta);

setOffsetN(ptr, getOffsetN(ptr) + d);

next = getNextPtr(next);

setOffsetP(next, getOffsetN(ptr));

**for** (**int** i = ptr; i < ptr + oldCs; i++)

memory[i - delta] = memory[i];

ptr -= delta;

**for** (**int** i = oldCs; i < getOffsetN(ptr); i++)

memory[ptr + i] = 0;

} **else** {

**int** t = getOffsetN(prev) + getOffsetN(ptr) + getOffsetN(next);

**int** oldCs = getOffsetN(ptr);

**int** oldPs = getOffsetP(ptr);

setOffsetN(ptr, t);

setOffsetP(ptr, getOffsetP(prev));

next = getNextPtr(next);

setOffsetP(next, t);

**for** (**int** i = ptr; i < ptr + oldCs; i++)

memory[i - oldPs] = memory[i];

ptr -= oldPs;

**for** (**int** i = oldCs; i < getOffsetN(ptr); i++)

memory[ptr + i] = 0;

}

} **else** {//найти подходящее место

**int** t = mem\_alloc(size);

**if** (t != -1) {

**for** (**int** i = 3; i < getOffsetN(ptr); i++)

memory[t + i] = memory[ptr + i];

mem\_free(ptr);

**return** t;

}

**return** ptr;

}

}

}

**return** ptr;

} **else** **if** (getHeader(getPrevPtr(ptr)) != 1) {//если с лева пусто

**int** prev = getPrevPtr(ptr);

**if** (getOffsetP(ptr) >= d) {//если достаточно места

**if** ((getOffsetP(ptr) - d) > 3) {//с созданием

**int** oldCs = getOffsetN(ptr);

setOffsetN(prev, getOffsetN(prev) - d);

setOffsetN(ptr, getOffsetN(ptr) + d);

setOffsetP(ptr, getOffsetP(ptr) - d);

setOffsetP(next, getOffsetP(next) + d);

**for** (**int** i = ptr; i < ptr + oldCs; i++)

memory[i - d] = memory[i];

ptr -= d;

**for** (**int** i = oldCs; i < getOffsetN(ptr); i++)

memory[ptr + i] = 0;

} **else** {

setHeader(prev, 1);

**int** cs = getOffsetN(prev);

**int** oldCs = getOffsetN(ptr);

setOffsetN(prev, getOffsetN(ptr) + cs);

setOffsetP(next, getOffsetN(ptr) + cs);

**for** (**int** i = ptr + 3; i < next; i++)

memory[i - cs] = memory[i];

**for** (**int** i = oldCs; i < getOffsetN(prev); i++)

memory[prev + i] = 0;

**return** prev;

}

} **else** {//найти подходящее место

**int** t = mem\_alloc(size);

**if** (t != -1) {

**for** (**int** i = 3; i < getOffsetN(ptr); i++)

memory[t + i] = memory[ptr + i];

mem\_free(ptr);

**return** t;

}

}

**return** ptr;

}

**return** ptr;

}

}

//Вивод виделеной памяти (заголовков)

**public** **void** blocks() {

**int** ptr = 0;

**int** t = 0;

**while** (getOffsetN(ptr) != 0) {

System.*out*.print(getHeader(ptr) + "| " + getOffsetN(ptr) + "| " + getOffsetP(ptr)

+ "| ");

ptr = getNextPtr(ptr);

**if** (t > memory.length) {

System.*out*.println();

**break**;

}

t++;

}

System.*out*.println(getHeader(ptr) + " " + getOffsetN(ptr) + " " + getOffsetP(ptr)

+ " ");

}

//проверка коректности заголовков

**public** **int** testPtrs() {

**int** ptr = getNextPtr(0);

**int** prev = 0;

**int** t = 0;

**while** (ptr != memory.length - 3) {

**if** ((getHeader(prev) == 0) && (getHeader(ptr) == 0)) {

System.*err*.println(" heads = 0 ");

**return** -1;

}

**if** (prev != getPrevPtr(ptr)) {

System.*err*.println(" prev ");

**return** -1;

}

**if** (t > memory.length) {

System.*err*.println(" t > length ");

**return** -1;

}

**if** (getOffsetN(ptr) < 4) {

System.*err*.println(" next < 4 ");

**return** -1;

}

ptr = getNextPtr(ptr);

prev = getNextPtr(prev);

t++;

}

**return** 0;

}

//заполнение виделеней памяти

**public** **void** fillBlock(**int** ptr) {

**int** maxValue = 100;

**for** (**int** i = 3; i < getOffsetN(ptr); i++)

memory[ptr + i] = (**int**) (Math.*random*() \* maxValue);

}

//Контрольная сума выделеной памяти

**public** **int** getControlSum(**int** ptr) {

**int** s = 0;

**for** (**int** i = 3; i < getOffsetN(ptr); i++)

s += memory[ptr + i];

**return** s;

}

**private** **int** getNextPtr(**int** ptr) {

**return** ptr + memory[ptr + 1];

}

**private** **int** getPrevPtr(**int** ptr) {

**return** ptr - memory[ptr + 2];

}

**private** **int** getOffsetN(**int** ptr) {

**return** memory[ptr + 1];

}

**private** **int** getOffsetP(**int** ptr) {

**return** memory[ptr + 2];

}

**private** **int** getHeader(**int** ptr) {

**return** memory[ptr];

}

**private** **void** setOffsetN(**int** ptr, **int** value) {

memory[ptr + 1] = value;

}

**private** **void** setOffsetP(**int** ptr, **int** value) {

memory[ptr + 2] = value;

}

**private** **void** setHeader(**int** ptr, **int** value) {

memory[ptr] = value;

}

**private** **int**[] memory;

}

**public** **class** test {

**public** **static** **void** main(String[] args) {

mem m = **new** mem(1000);

**int** maxSize = 100;

**int** n = 100;

**int**[] ptr = **new** **int**[n];

**int**[] controlSum = **new** **int**[n];

**for** (**int** i = 0; i < ptr.length; i++)

ptr[i] = -1;

**for** (**int** i = 0; i < 1000000; i++) {

**int** index = (**int**) (Math.*random*() \* (n - 1));

m.blocks();

**if** (ptr[index] == -1) {

**int** size = (**int**) (Math.*random*() \* maxSize);

ptr[index] = m.mem\_alloc(size);

m.fillBlock(ptr[index]);

controlSum[index] = m.getControlSum(ptr[index]);

} **else** {

**if** (Math.*random*() > 0.5) {

m.mem\_free(ptr[index]);

ptr[index] = -1;

} **else** {

**int** t = m.mem\_realloc(ptr[index],

(**int**) (Math.*random*() \* maxSize));

**if** (t != -1) {

**int** s = m.getControlSum(t);

**if** ((s != controlSum[index]) && (t != ptr[index])

&& (s > controlSum[index])) {

System.*err*.println("Error control sum");

**break**;

}

ptr[index] = t;

m.fillBlock(ptr[index]);

controlSum[index] = m.getControlSum(ptr[index]);

}

}

}

**if** (m.testPtrs() == -1)

**break**;

}

}

}