Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики і обчислювальної техніки

**Лабораторна робота №2**

**з курсу: «Системне програмування»**

*Виконав:*

студент групи ІС-72

Шумський В.П.

Залікова книжка №7232

*Перевірив:*

Сімоненко А.В.

Київ, 2020р.

**public class AllocTest {**

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

**mem m = new mem(4096, 256);**

**int maxSize = 350;**

**int n = 70;**

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

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

**ptr[i] = -1;**

**for (int i = 0; i < 10000000; i++) {**

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

**System.*out*.print(i + ") ");**

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

**int size = (int) (Math.*random*() \* maxSize) + 1;**

**System.*out*.print("alloc " + size + " ");**

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

**} else {**

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

**System.*out*.print("free ");**

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

**ptr[index] = -1;**

**} else {**

**int size = (int) (Math.*random*() \* maxSize) + 1;**

**System.*out*.print("realloc " + size + ' ');**

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

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

**ptr[index] = t;**

**}**

**}**

**}**

**if (!m. AllocTest ())**

**break;**

**else**

**System.*out*.println("OK");**

**}**

**if(m.test())**

**System.*out*.println("Test success");**

**}**

**}**

**public class Pages {**

**public Pages(int Pointer){**

**this.Pointer = Pointer;**

**this.next = null;**

**this.prev = null;**

**}**

**public Pages(Pages prev, int Pointer){**

**this.next = null;**

**this.Pointer = Pointer;**

**this.prev = prev;**

**}**

**public void setNext(Pages next) {**

**this.next = next;**

**}**

**public Pages getNext() {**

**return next;**

**}**

**public void setPrev(Pages prev) {**

**this.prev = prev;**

**}**

**public Pages getPrev() {**

**return prev;**

**}**

**public void setPointer(int pointer) {**

**Pointer = pointer;**

**}**

**public int getPointer() {**

**return Pointer;**

**}**

**private Pages next;**

**private Pages prev;**

**private int Pointer;**

**}**

**public class AllocMem {**

**public AllocMem (int size, int sizePage) {**

**this.sizePage = sizePage;**

**int t = size / sizePage;**

**t \*= 3;**

**t += size;**

**mem = new int[t];**

**correctPtr = new int[t / (sizePage + 3)];**

**t = minSize;**

**int c = 0;**

**while (t <= (sizePage >> 1)) {**

**c++;**

**t = t << 1;**

**}**

**ptrs = new int[c];**

**for (int i = 0; i < c; i++)**

**ptrs[i] = -1;**

**int i = 0;**

**int ptr = 0;**

**correctPtr[i] = ptr;**

**i++;**

**freeH = page(0);**

**int l = freeH;**

**ptr += sizePage + 3;**

**while (ptr < mem.length) {**

**correctPtr[i] = ptr;**

**i++;**

**setNextPtr(l, page(l, ptr));**

**l = getNextPtr(l);**

**ptr += sizePage + 3;**

**}**

**}**

**private int page(int i) {**

**setH(i, 0);**

**setCs(i, 0);**

**setPs(i, 0);**

**return i;**

**}**

**private int page(int i, int ptr) {**

**setH(ptr, 0);**

**setCs(ptr, 0);**

**setPs(ptr, Math.*abs*(ptr - i));**

**return ptr;**

**}**

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

**if (size <= (sizePage >> 1)) {**

**int s = minSize;**

**int i = 0;**

**while (s < size) {**

**i++;**

**s = s << 1;**

**}**

**if (ptrs[i] == -1) {**

**if (freeH == -1)**

**return -1;**

**ptrs[i] = freeH;**

**int ptr = freeH;**

**if (getCs(freeH) != 0)**

**freeH = getNextPtr(freeH);**

**else**

**freeH = -1;**

**if (freeH != -1)**

**setPs(freeH, 0);**

**setSize(ptr, s);**

**int t = 1;**

**t = t << (sizePage / s) - 1;**

**setBitMap(ptr, t);**

**setPs(ptr, 0);**

**return ptr + 3;**

**} else {**

**boolean b = false;**

**int l = ptrs[i];**

**while (l != -1) {**

**if (getCounter(l) < (sizePage / getSize(l))) {**

**b = true;**

**break;**

**}**

**if (getPs(l) == 0)**

**break;**

**l = getPrevPtr(l);**

**}**

**if (b) {**

**int ptr = l;**

**int t = 1;**

**t = t << (sizePage / s) - 1;**

**i = 0;**

**int map = getBitMap(ptr);**

**while ((map & t) != 0) {**

**i++;**

**t = t >> 1;**

**}**

**setBitMap(ptr, map | t);**

**ptr += (getSize(ptr) \* i) + 3;**

**return ptr;**

**} else {**

**if (freeH == -1)**

**return -1;**

**int p = freeH;**

**if (getCs(freeH) != 0) {**

**freeH = getNextPtr(freeH);**

**setPs(freeH, 0);**

**} else**

**freeH = -1;**

**p = page(p);**

**setPs(p, p - ptrs[i]);**

**ptrs[i] = p;**

**int ptr = p;**

**setSize(ptr, s);**

**int t = 1;**

**t = t << (sizePage / s) - 1;**

**setBitMap(ptr, t);**

**return ptr + 3;**

**}**

**}**

**} else {**

**int n = 1;**

**while (size > sizePage \* n)**

**n++;**

**int p = getNFreePages(n);**

**if (p == -1)**

**return -1;**

**int t = p;**

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

**if (getCs(t) == 0) {**

**t = -1;**

**break;**

**}**

**t = getNextPtr(t);**

**}**

**if (p == freeH) {**

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

**setPs(t, 0);**

**freeH = t;**

**} else {**

**setNextPtr(getPrevPtr(p), t);**

**setPrevPtr(t, getPrevPtr(p));**

**int ptr = p;**

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

**setH(ptr, 1);**

**setCs(ptr, sizePage + 3);**

**setPs(ptr, sizePage + 3);**

**ptr += sizePage + 3;**

**}**

**ptr -= sizePage + 3;**

**setPs(p, 0);**

**setCs(ptr, 0);**

**nBpages += n;**

**return p + 3;**

**}**

**int ptr = p;**

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

**setH(ptr, 1);**

**setCs(ptr, sizePage + 3);**

**setPs(ptr, sizePage + 3);**

**ptr += sizePage + 3;**

**}**

**setPs(p, 0);**

**setCs((ptr - sizePage) - 3, 0);**

**nBpages += n;**

**return p + 3;**

**}**

**}**

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

**int ptrS = ptr / (sizePage + 3);**

**ptrS \*= (sizePage + 3);**

**if (getSize(ptrS) != 1) {**

**if (getCounter(ptrS) > 1) {**

**int n = (ptr - ptrS) - 3;**

**n /= getSize(ptrS);**

**n = (sizePage / getSize(ptrS)) - n;**

**int t = 1;**

**t = t << n - 1;**

**setBitMap(ptrS, getBitMap(ptrS) ^ t);**

**} else {**

**int s = minSize;**

**int i = 0;**

**while (s < getSize(ptrS)) {**

**i++;**

**s = s << 1;**

**}**

**int p = ptrS;**

**if (p == ptrs[i]) {**

**if (getPs(ptrs[i]) == 0)**

**ptrs[i] = -1;**

**else**

**ptrs[i] = getPrevPtr(ptrs[i]);**

**} else {**

**if (getPs(p) == 0)**

**setPs(getNexti(i, p), 0);**

**else {**

**int t = getNexti(i, p);**

**setPs(t, t - getPrevPtr(p));**

**}**

**}**

**addFreePage(ptrS);**

**}**

**} else {**

**while (getCs(ptrS) != 0) {**

**addFreePage(ptrS);**

**ptrS += sizePage + 3;**

**nBpages--;**

**}**

**addFreePage(ptrS);**

**nBpages--;**

**}**

**}**

**public int getNexti(int i, int ptr) {**

**int p = ptrs[i];**

**while (getPrevPtr(p) != ptr)**

**p = getPrevPtr(p);**

**return p;**

**}**

**private int findFreePages(int ptrH, int ptrL, int n) {**

**int r = 0;**

**int l = 0;**

**if((ptrL+sizePage + 3)<mem.length)**

**ptrL += sizePage + 3;**

**while(getCs(ptrL) != 0){**

**if(getH(ptrL) != 0)**

**break;**

**r++;**

**ptrL += sizePage + 3;**

**}**

**if((ptrH-(sizePage + 3))>=0)**

**ptrH -= sizePage + 3;**

**while(getPs(ptrH) != 0){**

**if(getH(ptrH) != 0)**

**break;**

**l++;**

**ptrH -= sizePage + 3;**

**}**

**if (l + r >= n)**

**return r;**

**return -1;**

**}**

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

**int ptrS = ptr / (sizePage + 3);**

**ptrS \*= (sizePage + 3);**

**if (getSize(ptrS) != 1) {// ptr <= sizePage/2**

**if (size <= (sizePage >> 1)) { // new ptr <= sizePage/2**

**int s = minSize;**

**int i = 0;**

**while (s < size) {**

**i++;**

**s = s << 1;**

**}**

**if (ptrs[i] != -1)**

**if (getSize(ptrs[i]) == getSize(ptrS))**

**return ptr;**

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

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

**int tS = t / (sizePage + 3);**

**tS = tS \* (sizePage + 3);**

**int d = getSize(ptrS);**

**if (d > getSize(tS))**

**d = getSize(tS);**

**for (int j = 0; j < d; j++)**

**mem[t + j] = mem[ptr + j];**

**mem\_free(ptr);**

**}**

**return t;**

**} else { // new ptr > sizePage/2**

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

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

**for (int j = 0; j < getSize(ptrS); j++)**

**mem[t + j] = mem[ptr + j];**

**mem\_free(ptr);**

**}**

**return t;**

**}**

**} else {// ptr > sizePage./2**

**if (size <= (sizePage >> 1)) { // new ptr <= sizePage/2**

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

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

**int tS = t / (sizePage + 3);**

**tS = tS \* (sizePage + 3);**

**for (int j = 0; j < getSize(tS); j++)**

**mem[t + j] = mem[ptr + j];**

**mem\_free(ptr);**

**}**

**return t;**

**} else { // new ptr > sizePage/2**

**int n = 1;**

**while (size > sizePage \* n)**

**n++;**

**int cn = 1;**

**int tptr = ptrS;**

**while (getCs(tptr) != 0) {**

**cn++;**

**tptr += sizePage + 3;**

**}**

**if (n == cn)**

**return ptr;**

**if (cn > n) {**

**int d = cn - n;**

**tptr = ptrS;**

**for (int i = 0; i < cn - d; i++)**

**tptr += sizePage + 3;**

**setCs(tptr - (sizePage + 3), 0);**

**while (getCs(tptr) != 0) {**

**addFreePage(tptr);**

**nBpages--;**

**tptr += sizePage + 3;**

**}**

**addFreePage(tptr);**

**nBpages--;**

**return ptr;**

**} else {**

**int d = n - cn;**

**tptr = ptrS;**

**while (getCs(tptr) != 0)**

**tptr += sizePage + 3;**

**int r = findFreePages(ptrS, tptr, d);**

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

**if (r >= d) {**

**int p = freeH;**

**while (p != -1) {**

**if (p == tptr + sizePage + 3)**

**break;**

**p = getNextPtr(p);**

**}**

**int t = p;**

**for (int i = 0; i < d; i++)**

**t = getNextPtr(t);**

**if (p == freeH) {**

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

**setPs(t, 0);**

**freeH = t;**

**} else {**

**setNextPtr(getPrevPtr(p), t);**

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

**setPrevPtr(t, getPrevPtr(p));**

**}**

**setCs(tptr, sizePage + 3);**

**tptr = p;**

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

**setH(tptr, 1);**

**setCs(tptr, sizePage + 3);**

**setPs(tptr, sizePage + 3);**

**tptr += sizePage + 3;**

**}**

**setCs((tptr - sizePage) - 3, 0);**

**nBpages += d;**

**return ptr;**

**} else {**

**int p = freeH;**

**int t = p;**

**if (r != 0) {**

**p = freeH;**

**while (p != -1) {**

**if (p == tptr + sizePage + 3)**

**break;**

**p = getNextPtr(p);**

**}**

**t = p;**

**for (int i = 0; i < r; i++)**

**t = getNextPtr(t);**

**if (p == freeH) {**

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

**setPs(t, 0);**

**freeH = t;**

**} else {**

**setNextPtr(getPrevPtr(p),t);**

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

**setPrevPtr(t, getPrevPtr(p));**

**}**

**setCs(tptr, sizePage + 3);**

**tptr = p;**

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

**setH(tptr, 1);**

**setCs(tptr, sizePage + 3);**

**setPs(tptr, sizePage + 3);**

**tptr += sizePage + 3;**

**}**

**setCs((tptr - sizePage) - 3, 0);**

**}**

**tptr = ptrS;**

**p = freeH;**

**while (p != -1) {**

**if (p == tptr - (sizePage + 3))**

**break;**

**p = getNextPtr(p);**

**}**

**t = p;**

**for (int i = 0; i < d - r - 1; i++)**

**p = getPrevPtr(p);**

**if (p == freeH) {**

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

**setPs(t, 0);**

**freeH = t;**

**} else {**

**if(getCs(t) != 0){**

**t = getNextPtr(t);**

**setNextPtr(getPrevPtr(p), t);**

**setPrevPtr(t, getPrevPtr(p));**

**}**

**else**

**setCs(getPrevPtr(p), 0);**

**}**

**tptr = p;**

**for (int i = 0; i < d - r; i++) {**

**setH(tptr, 1);**

**setCs(tptr, sizePage + 3);**

**setPs(tptr, sizePage + 3);**

**tptr += sizePage + 3;**

**}**

**tptr = p;**

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

**for (int j = 0; j < sizePage + 3; j++)**

**mem[tptr + j] = mem[ptrS + j];**

**tptr += sizePage + 3;**

**ptrS += sizePage + 3;**

**}**

**setCs(tptr, sizePage + 3);**

**nBpages += d;**

**return p;**

**}**

**} else {**

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

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

**tptr = ptrS;**

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

**for (int j = 0; j < getCs(ptrS); j++)**

**mem[t + j] = mem[ptrS + j];**

**ptrS += sizePage + 3;**

**}**

**mem\_free(tptr);**

**}**

**return t;**

**}**

**}**

**}**

**}**

**}**

**private int getNFreePages(int n) {**

**int p = freeH;**

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

**if (n == 1)**

**return p;**

**while (getCs(p) != 0) {**

**boolean find = true;**

**for (int i = 0; i < n; i++)**

**if (getCs(p) == 0) {**

**find = false;**

**break;**

**} else {**

**if (p + sizePage + 3 != getNextPtr(p)) {**

**find = false;**

**break;**

**} else**

**p = getNextPtr(p);**

**}**

**if (find) {**

**for (int i = 0; i < n; i++)**

**p = getPrevPtr(p);**

**return p;**

**}**

**if (getCs(p) != 0)**

**p = getNextPtr(p);**

**else**

**break;**

**}**

**}**

**return -1;**

**}**

**public boolean testPtr(int ptr) {**

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

**if (ptr == correctPtr[i])**

**return true;**

**System.*err*.println(" test ptr ");**

**return false;**

**}**

**private void addFreePage(int ptr) {**

**if (freeH == -1) {**

**freeH = page(ptr);**

**} else {**

**setH(ptr, 0);**

**int p = freeH;**

**while (p < ptr)**

**if (getCs(p) == 0)**

**break;**

**else**

**p = getNextPtr(p);**

**setNextPtr(ptr, p);**

**if (ptr < p) {**

**if (p == freeH) {**

**freeH = ptr;**

**setPs(ptr, 0);**

**} else {**

**setPrevPtr(ptr, getPrevPtr(p));**

**setNextPtr(getPrevPtr(p), ptr);**

**}**

**setPrevPtr(p, ptr);**

**} else {**

**ptr = page(ptr);**

**setNextPtr(p, ptr);**

**setPrevPtr(ptr, p);**

**}**

**}**

**}**

**public boolean test() {**

**if (nBpages < 0) {**

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

**return false;**

**}**

**if (freeH != -1)**

**if (getPs(freeH) != 0) {**

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

**return false;**

**}**

**int ptr = 0;**

**int n = 0;**

**while (ptr < mem.length) {**

**n++;**

**ptr += sizePage + 3;**

**}**

**int cn = 0;**

**int p = freeH;**

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

**while (p != -1) {**

**if (!testPtr(p))**

**return false;**

**pointers[cn] = p;**

**cn++;**

**if (getCs(p) == 0)**

**break;**

**p = getNextPtr(p);**

**if (p != -1)**

**if (getPrevPtr(p) > p) {**

**System.*err*.println(" free pages ");**

**return false;**

**}**

**}**

**for (int i = 0; i < cn; i++)**

**for (int j = i + 1; j < cn; j++)**

**if (pointers[i] == pointers[j]) {**

**System.*err*.println(" ptr=ptr ");**

**return false;**

**}**

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

**p = ptrs[i];**

**while (p != -1) {**

**if (!testPtr(p))**

**return false;**

**if (getCounter(p) < 1)**

**break;**

**cn++;**

**if (getPs(p) == 0)**

**break;**

**p = getPrevPtr(p);**

**}**

**}**

**cn += nBpages;**

**if (cn != n) {**

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

**return false;**

**}**

**return true;**

**}**

**private void setSize(int ptr, int size) {**

**mem[ptr] = size;**

**}**

**private int getSize(int ptr) {**

**return mem[ptr];**

**}**

**private void setBitMap(int ptr, int map) {**

**mem[ptr + 1] = map;**

**}**

**private int getBitMap(int ptr) {**

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

**}**

**private int getCounter(int ptr) {**

**int t = 1;**

**int c = 0;**

**while (t != 0) {**

**if ((getBitMap(ptr) & t) > 0)**

**c++;**

**t = t << 1;**

**}**

**return c;**

**}**

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

**mem[ptr + 1] = value;**

**}**

**private int getCs(int ptr) {**

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

**}**

**private int getPs(int ptr) {**

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

**}**

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

**mem[ptr + 2] = value;**

**}**

**private int getH(int ptr) {**

**return mem[ptr];**

**}**

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

**mem[ptr] = value;**

**}**

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

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

**}**

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

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

**}**

**private void setNextPtr(int ptr, int next) {**

**mem[ptr + 1] = Math.*abs*(ptr - next);**

**}**

**private void setPrevPtr(int ptr, int pr) {**

**mem[ptr + 2] = Math.*abs*(ptr - pr);**

**}**

**private int[] correctPtr;**

**private int mem[];**

**private int[] ptrs;**

**private int freeH;**

**private int nBpages = 0;**

**private int sizePage;**

**private int minSize = 16;**

**}**