**Homework 3**

|  |
| --- |
| 그림입니다. 원본 그림의 이름: YU_UI_RGB-10.png 원본 그림의 크기: 가로 2256pixel, 세로 3047pixel 프로그램 이름 : Adobe ImageReady |

|  |  |
| --- | --- |
| 과목명 | 객체지향프로그래밍과자료구조 |
| 교수님 | 김영탁 교수님 |
| 이 름 | 김주환 |
| 학 번 | 21812158 |
| 일 자 | 2021.09.22.수 |

1. 정수 데이터를 1,000,000개 담을 수 있는 동적 배열 생성 및 정렬 알고리즘 구현

|  |
| --- |
| /\* main.cpp \*/  /\* Description  \* 원하는 크기의 무작위 배열생성  \* 배열의 특성(최대, 최소, 평균 등등)  \* 선택/퀵 정렬 및 소요 시간 측정  \* Programmed by J. H. Kim  \* Last updated : 2021-09-22 \*/  #include <iostream>  #include <fstream>  #include <Windows.h>  #include <time.h>  #include "BigArray.h"  using namespace std;  #define ELEMENTS\_PER\_LINE 10  #define SAMPLE\_LINES 3  void main()  {  LARGE\_INTEGER freq, t1, t2;  LONGLONG t\_diff;  double elapsed\_t;  QueryPerformanceFrequency(&freq); // 주파수  ofstream fout;  fout.open("output.txt");  if (fout.fail())  {  cout << "Error in opening output.txt !!"  << endl;  exit;  }  int base\_offset = 0;  int big\_rand\_size = 1000000; // 배열 크기 1,000,000  base\_offset = -big\_rand\_size / 2; // 오프셋  BigArray ba\_2(big\_rand\_size);  fout << endl << "Generating big rand array of "  << ba\_2.size() << " elements with base\_offset " << base\_offset << " ... "  << endl;  ba\_2.genBigRandArray(base\_offset); // 배열 생성  ba\_2.fprintSample(fout, ELEMENTS\_PER\_LINE, SAMPLE\_LINES);  ba\_2.fprintStatistics(fout);  QueryPerformanceCounter(&t1);  ba\_2.quick\_sort(); // 퀵소트 및 시간 측정  QueryPerformanceCounter(&t2);  t\_diff = t2.QuadPart - t1.QuadPart;  elapsed\_t = (double)t\_diff / freq.QuadPart;  fout << "Quick sort took " << elapsed\_t \* 1000 << "[milliseconds]" << endl;  ba\_2.fprintSample(fout, ELEMENTS\_PER\_LINE, SAMPLE\_LINES);  ba\_2.suffle();  QueryPerformanceCounter(&t1);  ba\_2.selection\_sort(); // 선택 정렬 및 시간 측정  QueryPerformanceCounter(&t2);  t\_diff = t2.QuadPart - t1.QuadPart;  elapsed\_t = (double)t\_diff / freq.QuadPart;  fout << "Selection sort took " << elapsed\_t \* 1000 << " [milliseconds]" << endl;  ba\_2.fprintSample(fout, ELEMENTS\_PER\_LINE, SAMPLE\_LINES);  fout.close();  } |
| /\* BigArray.h \*/  #ifndef BIG\_ARRAY\_H  #define BIG\_ARRAY\_H  #include <iostream>  #include <fstream>  using namespace std;  typedef struct  {  int min;  int max;  double avg; // average  double medi; // median  double var; // variance  double std\_dev; // standard deviation  } ArrayStatistics;  class BigArray  {  public:  BigArray(int n); // constructor  ~BigArray(); // destructor  void genBigRandArray(int base\_offset);  int size() { return num\_elements; }  void suffle();  void selection\_sort();  void quick\_sort();  void getStatistics(ArrayStatistics&);  void fprintStatistics(ostream& fout);  void fprintBigArray(ostream& fout, int  elements\_per\_line);  void fprintSample(ostream& fout, int  elements\_per\_line, int num\_sample\_lines);  private:  int\* big\_array;  int num\_elements;  };  #endif |
| /\* BigArray.cpp \*/  #include <iostream>  #include <stdlib.h>  #include <time.h>  #include <iomanip>  #include "BigArray.h"  using namespace std;  BigArray::BigArray(int n) // constructor  :num\_elements(n)  {  big\_array = (int\*) new int[num\_elements];  if (big\_array == NULL)  {  cout << "Error in creation of dynamic array of size(" << num\_elements  << ") !!" << endl;  exit;  }  }  BigArray::~BigArray() // destructor  {  if (big\_array != NULL)  delete[] big\_array;  }  void BigArray::genBigRandArray(int base\_offset)  {  char\* flag;  int count = 0;  int rand\_h, rand\_l, big\_rand, biased\_big\_rand;  srand(time(0));  flag = (char\*) new char[num\_elements];  while (count < num\_elements)  {  rand\_h = rand();  rand\_l = rand();  big\_rand = ((long)rand\_h << 15) | rand\_l;  big\_rand = big\_rand % num\_elements;  if (flag[big\_rand] == 1)  continue;  else  {  flag[big\_rand] = 1;  biased\_big\_rand = big\_rand + base\_offset;  big\_array[count] = biased\_big\_rand;  count++;  }  }  delete[] flag;  }  void BigArray::getStatistics(ArrayStatistics  & stats)  {  int min = INT\_MAX;  int max = INT\_MIN;  double mean = 0.0;  double sum = 0.0;  double medi = 0.0;  double sq\_sum\_avg = 0.0;  double diff\_sq\_sum = 0.0;  double var, std\_dev;  int element;  for (int i = 0; i < num\_elements; i++)  {  element = big\_array[i];  sum += element;  if (element > max)  max = element;  if (element < min)  min = element;  }  mean = sum / (double)num\_elements;  diff\_sq\_sum = 0.0;  for (int i = 0; i < num\_elements; i++)  {  element = big\_array[i];  diff\_sq\_sum +=  (element - mean) \* (element - mean);  }  var = diff\_sq\_sum / (double)num\_elements;  std\_dev = sqrt(var);  int indx;  quick\_sort();  indx = int(num\_elements / 2) - 1;  if (num\_elements % 2) {  medi = big\_array[indx + 1];  }  else {  medi = (big\_array[indx] + big\_array[indx + 1]) / 2.0;  }  suffle();  stats.min = min;  stats.max = max;  stats.avg = mean;  stats.medi = medi;  stats.var = var;  stats.std\_dev = std\_dev;  }  void BigArray::fprintStatistics(ostream& fout)  {  ArrayStatistics stats;  fout.setf(ios::fixed);  fout.setf(ios::showpoint);  fout.precision(2);  getStatistics(stats);  fout << "Statistics: " << endl;  fout << " min (" << stats.min << "), max ("  << stats.max << "), avg (" << stats.avg;  fout << "), medi (" << stats.medi << "), var (" << stats.var << "), std\_dev ("  << stats.std\_dev << ")" << endl << endl;  }  void BigArray::suffle()  {  srand(time(0));  int index1, index2;  int rand\_1, rand\_2;  int temp;  for (int i = 0; i < num\_elements; i++)  {  rand\_1 = rand();  rand\_2 = rand();  index1 = ((rand\_1 << 15) | rand\_2)  % num\_elements;  rand\_1 = rand();  rand\_2 = rand();  index2 = ((rand\_1 << 15) | rand\_2)  % num\_elements;  temp = big\_array[index1];  big\_array[index1] = big\_array[index2];  big\_array[index2] = temp;  }  }  void BigArray::selection\_sort()  {  int min; // index of the element with minimum value  double minValue; // minimum value  for (int i = 0; i < num\_elements - 1; i++)  {  min = i;  minValue = big\_array[i];  for (int j = i + 1; j < num\_elements; j++)  {  if (minValue > big\_array[j])  {  min = j;  minValue = big\_array[j];  }  }  if (min != i) // if a smaller element is found, then swap  {  /\* minValue is dA[min] \*/  big\_array[min] = big\_array[i];  big\_array[i] = minValue;  }  } // end for  }  int \_partition(int\* array, int size, int left, int right,  int pivotIndex, int level)  {  int pivotValue; // pivot value  int newPI; // store index  double temp;  int i;  #ifdef DEBUG\_QUICKSORT  for (i = 0; i < level; i++) // put indentation  fout << " ";  fout << " Partition (left " << left << ", right "  << right << ", pivotIdex "  << pivotIndex << "(pV:"  << array[pivotIndex]  << ") ) =>";  #endif  pivotValue = array[pivotIndex];  temp = array[pivotIndex];  array[pivotIndex] = array[right];  array[right] = temp; // Move pivot to end  newPI = left;  for (i = left; i <= (right - 1); i++) {  if (array[i] <= pivotValue) {  temp = array[i];  array[i] = array[newPI];  array[newPI] = temp;  newPI = newPI + 1;  }  }  // swap array[newPI] and array[right];  // Move pivot to its final place  temp = array[newPI];  array[newPI] = array[right];  array[right] = temp;  return newPI;  }  void \_quick\_sort(int\* array, int size, int left, int right, int level)  {  int pI, newPI; // pivot index  if (left >= right) {  return;  }  else if (left < right)  { // subarray of 0 or 1 elements already sorted  //select a pI (pivotIndex) in the range left ≤ pI ≤ right  pI = (left + right) / 2;  }  newPI = \_partition(array, size, left, right, pI, level);  if (left < (newPI - 1))  {  \_quick\_sort(array, size, left, newPI - 1, level + 1);  // recursively sort elements on the left of pivotNewIndex  }  if ((newPI + 1) < right)  {  \_quick\_sort(array, size, newPI + 1, right, level + 1);  // recursively sort elements on the right of pivotNewIndex  }  } // end \_quick\_sort()  void BigArray::quick\_sort()  {  int pI, newPI; // pivot index  \_quick\_sort(big\_array, num\_elements, 0, num\_elements - 1, 0);  }  void BigArray::fprintBigArray(ostream& fout, int elements\_per\_line)  {  int count = 0;  while (count < num\_elements)  {  fout << setw(5) << big\_array[count];  count++;  if (count % elements\_per\_line == 0)  fout << endl;  }  fout << endl;  }  void BigArray::fprintSample(ostream& fout,  int elements\_per\_line,  int num\_sample\_lines)  {  int last\_block\_start;  int count = 0;  for (int i = 0; i < num\_sample\_lines; i++)  {  for (int j = 0; j < elements\_per\_line; j++)  {  if (count > num\_elements)  {  fout << endl;  return;  }  fout << setw(10) << big\_array[count];  count++;  }  fout << endl;  }  if (count < (num\_elements - elements\_per\_line \* num\_sample\_lines))  count = num\_elements - elements\_per\_line \* num\_sample\_lines;  fout << " . . . . . " << endl << endl;  for (int i = 0; i < num\_sample\_lines; i++)  {  for (int j = 0; j < elements\_per\_line; j++)  {  if (count > num\_elements)  {  fout << endl;  return;  }  fout << setw(10)  << big\_array[count];  count++;  }  fout << endl;  }  fout << endl;  } |
|  |

2. 행렬 데이터를 파일로부터 세 번 입력, class Mtrx 객체에 저장, 첫 두 행렬의 덧셈, 뺄셈 계산을 실행, 첫째와 셋째 행렬의 곱셈 계산을 각각 실행, 결과를 출력 파일에 출력

|  |
| --- |
| /\* main.cpp \*/  /\* Description  \* 2차원 배열로 행렬을 구현  \* 기본연산 구현  \* Programmed by J. H. Kim  \* Last updated : 2021-09-22 \*/  #include <iostream>  #include <fstream>  #include "Class\_Mtrx.h"  using namespace std;  void main()  {  ifstream fin;  ofstream fout;  fin.open("input.txt");  if (fin.fail())  {  cout << "Error in opening Matrix\_5x5\_data.txt !!" << endl;  exit;  }  fout.open("output.txt");  if (fout.fail())  {  cout << "Error in opening Matrix\_operations\_results.txt !!" << endl;  exit;  }  // 기본 행렬 데이터 불러오기  Mtrx mtrxA(fin);  int n\_row = mtrxA.getN\_row();  int n\_col = mtrxA.getN\_col();  Mtrx mtrxB(fin);  Mtrx mtrxC(fin);  // 행렬 덧셈  Mtrx mtrxD(mtrxA.getN\_row(), mtrxB.getN\_col());  mtrxD = mtrxA.add(mtrxB);  mtrxD.setName("Mtrx\_SUM = mtrxA.add(mtrxB)");  mtrxD.fprintMtrx(fout);  // 행렬 뺄셈  Mtrx mtrxE(mtrxA.getN\_row(), mtrxB.getN\_col());  mtrxE = mtrxA.sub(mtrxB);  mtrxE.setName("Mtrx\_SUB = mtrxA.sub(mtrxB)");  mtrxE.fprintMtrx(fout);  // 행렬 곱셈  Mtrx mtrxF(mtrxA.getN\_row(), mtrxC.getN\_col());  mtrxF = mtrxA.multiply(mtrxC);  mtrxF.setName("Mtrx\_MUL = mtrxA.multiply(mtrxC)");  mtrxF.fprintMtrx(fout);  fout.close();  } // end of main() |
| /\* Class\_Mtrx.h \*/  #ifndef MTRX\_H  #define MTRX\_H  #include <iostream>  #include <fstream>  using namespace std;  #define MAX\_SIZE 100  class Mtrx {  public:  Mtrx(int num\_row, int num\_col); // 클래스 생성자  Mtrx(double\* dA, int num\_data, int num\_row, int num\_col); // 생성자  Mtrx(istream& fin); // 생성자  ~Mtrx(); // destructor  int getN\_row() { return n\_row; } // 열 정보 반환  int getN\_col() { return n\_col; } // 행 정보 반환  void fprintMtrx(ostream& fout); // 포멧에 맞춰 출력  void setName(string nm) { name = nm; }; // 이름 설정  string getName() { return name; }; // 이름 반환  Mtrx add(const Mtrx&); // 덧셈 연산  Mtrx sub(const Mtrx&); // 뺄셈 연산  Mtrx multiply(const Mtrx&); // 곱셈 연산  private:  string name;  int n\_row;  int n\_col;  double\*\* dM;  };  #endif |
| /\* Class\_Mtrx.cpp \*/  #include "Class\_Mtrx.h"  #include <iostream>  #include <iomanip>  using namespace std;  typedef double\* DBLPTR;  Mtrx::Mtrx(int num\_row, int num\_col)  {  int i, j;  //cout <<"Mtrx constructor (int size: "  // << size << ")₩n";  n\_row = num\_row;  n\_col = num\_col;  dM = new DBLPTR[n\_row];  for (i = 0; i < n\_row; i++)  {  dM[i] = new double[n\_col];  }  for (i = 0; i < n\_row; i++) {  for (j = 0; j < n\_col; j++) {  dM[i][j] = 0.0;  }  }  // cout <<"End of Mtrx constructor...₩n";  }  Mtrx::~Mtrx()  {  // cout << "destructor of Mtrx ("  // << name << ")" << endl;  // 주석처리하지 않으면 중간에 소멸자를 호출하여 기능을 수행하지 못함  /\*  for (int i=0; i<n\_row; i++)  delete [] dM[i];  delete [] dM;  \*/  }  Mtrx::Mtrx(istream& fin)  {  // DBLPTR \*dM; /\* defined in class, as private data member  int i, j, size\_row, size\_col, num\_data, cnt;  double d;  //cout <<"Mtrx constructor (double \*\*dA, int size: " << size << ")₩n";  fin >> size\_row >> size\_col;  n\_row = size\_row;  n\_col = size\_col;  dM = new DBLPTR[n\_row];  for (i = 0; i < n\_row; i++)  {  dM[i] = new double[n\_col];  }  for (i = 0; i < n\_row; i++) {  for (j = 0; j < n\_col; j++) {  if (fin.eof())  dM[i][j] = 0.0;  else  {  fin >> d;  dM[i][j] = d;  }  }  }  //cout <<"End of Mtrx constructor... ₩n";  }  #define SETW 6  void Mtrx::fprintMtrx(ostream& fout)  {  unsigned char a6 = 0xA6, a1 = 0xA1, a2 = 0xA2;  unsigned char a3 = 0xA3, a4 = 0xA4, a5 = 0xA5;  fout << name << " =\n";  for (int i = 0; i < n\_row; i++) {  for (int j = 0; j < n\_col; j++)  {  fout.setf(ios::fixed);  fout.precision(1);  if ((i == 0) && (j == 0))  fout << a6 << a3 << setw(SETW) << dM[i][j];  else if ((i == 0) && (j == (n\_col - 1)))  fout << setw(SETW) << dM[i][j] << a6 << a4;  else if ((i > 0) && (i < (n\_row - 1)) && (j == 0))  fout << a6 << a2 << setw(SETW) << dM[i][j];  else if ((i > 0) && (i < (n\_row - 1)) && (j == (n\_col - 1)))  fout << setw(SETW) << dM[i][j] << a6 << a2;  else if ((i == (n\_row - 1)) && (j == 0))  fout << a6 << a6 << setw(SETW) << dM[i][j];  else if ((i == (n\_row - 1)) && (j == (n\_col - 1)))  fout << setw(SETW) << dM[i][j] << a6 << a5;  else  fout << setw(SETW) << dM[i][j];  }  fout << endl;  }  fout << endl;  }  Mtrx Mtrx::add(const Mtrx& mA)  {  int i, j;  Mtrx mR(n\_row, n\_col);  mR.setName("R");  for (i = 0; i < n\_row; i++) {  for (j = 0; j < n\_col; j++) {  mR.dM[i][j] = dM[i][j] + mA.dM[i][j];  }  }  return mR;  }  Mtrx Mtrx::sub(const Mtrx& mA)  {  int i, j;  Mtrx mR(n\_row, n\_col);  mR.setName("R");  for (i = 0; i < n\_row; i++) {  for (j = 0; j < n\_col; j++) {  mR.dM[i][j] = dM[i][j] - mA.dM[i][j];  }  }  return mR;  }  Mtrx Mtrx::multiply(const Mtrx& mA)  {  int i, j, k;  Mtrx mR(n\_row, mA.n\_col);  mR.setName("R");  for (i = 0; i < n\_row; i++) {  for (j = 0; j < mA.n\_col; j++) {  mR.dM[i][j] = 0.0;  for (k = 0; k < n\_col; k++) {  mR.dM[i][j] += dM[i][k] \* mA.dM[k][j];  }  }  }  return mR;  } |
|  |
|  |