**Homework 7**

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

|  |  |
| --- | --- |
| 과목명 | 프로그래밍 언어 |
| 교수님 | 김영탁 교수님 |
| 이 름 | 김주환 |
| 학 번 | 21812158 |
| 일 자 | 2021.04.19.월 |

1. **함수의 인수 전달 방식에 대한 비교**

* 두개의 double 자료형 데이터를 함수의 인수로 전달, 평균값 계산 후 double 자료형으로 반환하는 간단한 average() 함수를 call-by-value, call-by-pointer, call-by-reference 방식으로 각각 구현하고, 정상적으로 실행되는 것을 확인하라.

/\*

\* 파일명 : call\_by.cpp

\* 목적 및 기본 기능

\* 2개의 실수 데이터를 함수의 인수로 전달하고, 그 평균값을 계산하여 실수 자료형으로 반환하는

\* 간단한 average() 함수를 call-by-value, call-by-pointer, call-by-reference 방식으로 각각 구현하라

\* 작성자 : 김주환(21812158)

\* 작성일 : 2021년 4월 19일

\*/

#include <stdio.h>

double avg\_v(double x, double y);

void avg\_p(double\* x, double\* y, double\* avg);

void avg\_r(double& x, double& y, double& avg);

int main(void) {

double x = 10.0, y = 20.0;

double avg = 0.0;

avg = avg\_v(x, y);

printf("Average of call-by-value method : %lf\n", avg);

avg\_p(&x, &y, &avg);

printf("Average of call-by-pointer method : %lf\n", avg);

avg\_r(x, y, avg);

printf("Average of call-by-reference method : %lf\n", avg);

return 0;

}

double avg\_v(double x, double y) {

double avg = 0.0;

avg = (x + y) / 2.0;

return avg;

}

void avg\_p(double\* x, double\* y, double\* avg) {

\*avg = (\*x + \*y) / 2.0;

}

void avg\_r(double& x, double& y, double& avg) {

avg = (x + y) / 2.0;

}

텍스트이(가) 표시된 사진

자동 생성된 설명

* 이 세가지 함수 인수 전달 방법의 장점과 단점을 비교하여 설명하라.

값에 의한 호출과 나머지 호출 방식을 비교할 경우 호출한 변수를 변경할 수 있는지가 가장 큰 차이점이다. 값에 의한 호출은 변경할 수 없다. 함수 차원에서 호출한 변수의 값을 변경할 수 있다는 말은 하나의 함수에서 여러 개의 값을 반환할 수 있다는 말과 같다.

1. **파일 입력, 2차원 배열의 동적 생성, 행렬의 덧셈, 뺄셈, 곱셈 연산**

/\*

\* 파일명 : main\_dynammic\_2D\_calc.cpp

\* 목적 및 기본 기능

\* 2차원 배열을 동적 할당을 통해 생성하고 행렬 덧셈, 뺄셈, 곱셈, 평균을 구하는 계산

\* 을 실행하고 실행창과 파일에 출력한다.

\* 작성자 : 김주환(21812158)

\* 작성일 : 2021년 4월 19일

\*/

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include "matrix.h"

#define MTRXINPUTDATA "input\_data.txt"

int main(void) {

FILE\* fin;

double\*\* dma = NULL, \*\* dmb = NULL, \*\* dmc = NULL;

int a\_row\_size = 0, a\_col\_size = 0;

int b\_row\_size = 0, b\_col\_size = 0;

int c\_row\_size = 0, c\_col\_size = 0;

int temp = 0;

fin = fopen(MTRXINPUTDATA, "r"); // 파일 읽기 모드로 열기

if (fin == NULL) {

printf("Error in openning input.txt file !!\n");

exit(-1);

}

dma = input\_mtrx(fin, dma, &a\_row\_size, &a\_col\_size);

printf("Input Matrix\_A (%d x %d) : \n", a\_row\_size, a\_col\_size);

print\_mtrx(dma, a\_row\_size, a\_col\_size);

printf("\n");

dmb = input\_mtrx(fin, dmb, &b\_row\_size, &b\_col\_size);

printf("Input Matrix\_B (%d x %d) : \n", b\_row\_size, b\_col\_size);

print\_mtrx(dmb, b\_row\_size, b\_col\_size);

printf("\n");

c\_row\_size = a\_row\_size;

c\_col\_size = b\_col\_size;

dmc = input\_mtrx(fin, dmc, &c\_row\_size, &c\_col\_size);

dmc = add\_mtrx(dma, dmb, dmc, a\_row\_size, a\_col\_size);

printf("Matrix\_C (%d x %d) = Matrix\_A + Matrix\_B :\n", c\_row\_size, c\_col\_size);

print\_mtrx(dmc, c\_row\_size, c\_col\_size);

printf("\n");

dmc = sub\_mtrx(dma, dmb, dmc, a\_row\_size, a\_col\_size);

printf("Matrix\_D (%d x %d) = Matrix\_A - Matrix\_B :\n", c\_row\_size, c\_col\_size);

print\_mtrx(dmc, c\_row\_size, c\_col\_size);

printf("\n");

temp = c\_row\_size;

dmc = multiply\_mtrx(dma, dmb, dmc, c\_row\_size, c\_col\_size, temp);

printf("Matrix\_C (%d x %d) = Matrix\_A x Matrix\_B :\n", c\_row\_size, c\_col\_size);

print\_mtrx(dmc, c\_row\_size, c\_col\_size);

printf("\n");

delete\_double\_mtrx(dma, a\_row\_size, a\_col\_size);

delete\_double\_mtrx(dmb, b\_row\_size, b\_col\_size);

delete\_double\_mtrx(dmc, c\_row\_size, c\_col\_size);

fclose(fin);

return 0;

}

// matrix.cpp

#ifndef M // 헤더 중복 방지

#define M

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include "matrix.h"

#define MTRXINOUTDATA "input\_data.txt"

double\*\* input\_mtrx(FILE\* fin, double\*\* dm, int\* row\_size, int\* col\_size) {

double d = 0.0;

int temp;

fscanf(fin, "%d %d", row\_size, col\_size);

dm = (double\*\*)calloc(\*row\_size, sizeof(double\*));

for (int i = 0; i < \*row\_size; i++) {

dm[i] = (double\*)calloc(\*col\_size, sizeof(double));

}

// if (fscanf(fin, "%d", &temp) == EOF) return dm;

for (int i = 0; i < \*row\_size; i++) {

for (int j = 0; j < \*col\_size; j++) {

if (fscanf(fin, "%lf", &d) != EOF) dm[i][j] = d;

}

}

return dm;

}

void print\_mtrx(double\*\* dm, int row\_size, int col\_size) {

unsigned char a6 = 0xA6, a1 = 0xA1, a2 = 0xA2;

unsigned char a3 = 0xA3, a4 = 0xA4, a5 = 0xA5;

for (int i = 0; i < row\_size; i++) {

for (int j = 0; j < col\_size; j++) {

if ((i == 0) && (j == 0)) {

printf("%c%c%7.2lf", a6, a3, dm[i][j]);

}

else if ((i == 0) && j == (col\_size - 1)) {

printf("%7.2lf%c%c", dm[i][j], a6, a4);

}

else if ((i > 0) && (i < row\_size - 1) && (j == 0)) {

printf("%c%c%7.2lf", a6, a2, dm[i][j]);

}

else if ((i > 0) && (i < row\_size - 1) && (j == (col\_size - 1))) {

printf("%7.2lf%c%c", dm[i][j], a6, a2);

}

else if ((i == row\_size - 1) && (j == 0)) {

printf("%c%c%7.2lf", a6, a6, dm[i][j]);

}

else if ((i == row\_size - 1) && (j == (col\_size - 1))) {

printf("%7.2lf%c%c", dm[i][j], a6, a5);

}

else {

printf("%7.2lf", dm[i][j]);

}

}

printf("\n");

}

}

// matrix.cpp (2)

void delete\_double\_mtrx(double\*\* dm, int row\_size, int col\_size) {

for (int i = 0; i < row\_size; i++) free(dm[i]);

free(dm);

}

double\*\* add\_mtrx(double\*\* dm1, double\*\* dm2, double\*\* dm3, int row\_size, int col\_size) {

for (int i = 0; i < row\_size; i++) {

for (int j = 0; j < col\_size; j++) {

dm3[i][j] = 0;

dm3[i][j] = dm1[i][j] + dm2[i][j];

}

}

return dm3;

}

double\*\* sub\_mtrx(double\*\* dm1, double\*\* dm2, double\*\* dm3, int row\_size, int col\_size) {

for (int i = 0; i < row\_size; i++) {

for (int j = 0; j < col\_size; j++) {

dm3[i][j] = 0;

dm3[i][j] = dm1[i][j] - dm2[i][j];

}

}

return dm3;

}

double\*\* multiply\_mtrx(double\*\* dm1, double\*\* dm2, double\*\* dm3, int row\_size, int col\_size, int temp) {

for (int i = 0; i < row\_size; i++) {

for (int j = 0; j < col\_size; j++) {

dm3[i][j] = 0;

for (int k = 0; k < temp; k++) {

dm3[i][j] += dm1[i][k] \* dm2[k][j];

}

}

}

return dm3;

}

#endif

테이블이(가) 표시된 사진

자동 생성된 설명

5 5

1.1 2.2 3.3 4.4 5.5

6.6 7.7 8.8 9.9 10.0

11.1 12.2 13.3 14.4 15.5

16.6 17.7 18.8 19.9 20.0

21.1 22.2 23.3 24.4 25.5

5 5

1.1 0.0 0.0 0.0 0.0

0.0 2.2 0.0 0.0 0.0

0.0 0.0 3.3 0.0 0.0

0.0 0.0 0.0 4.4 0.0

0.0 0.0 0.0 0.0 5.5

// matrix.h

#ifndef M // 헤더 중복 방지

#define M

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#define MTRXINPUTDATA "input\_data.txt"

double\*\* input\_mtrx(FILE\* fin, double\*\* dm, int\* row\_size, int\* col\_size);

void print\_mtrx(double\*\* dm, int row\_size, int col\_size);

void delete\_double\_mtrx(double\*\* dm, int row\_size, int col\_size);

double\*\* add\_mtrx(double\*\* dm1, double\*\* dm2, double\*\* dm3, int row\_size, int col\_size);

double\*\* sub\_mtrx(double\*\* dm1, double\*\* dm2, double\*\* dm3, int row\_size, int col\_size);

double\*\* multiply\_mtrx(double\*\* dm1, double\*\* dm2, double\*\* dm3, int row\_size, int col\_size, int temp);

#endif