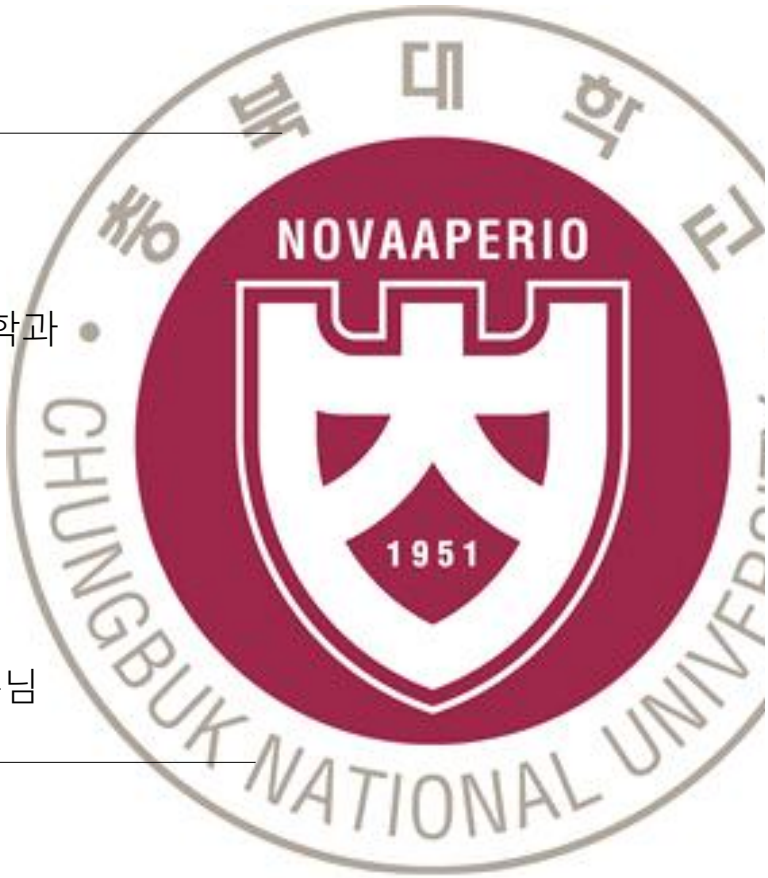
CHUNGBUK NATIONAL UNIVERSITY

REPORT

객체지향 프로그래밍 실습#2



제출일 2021.03.22

학과 소프트웨어학과

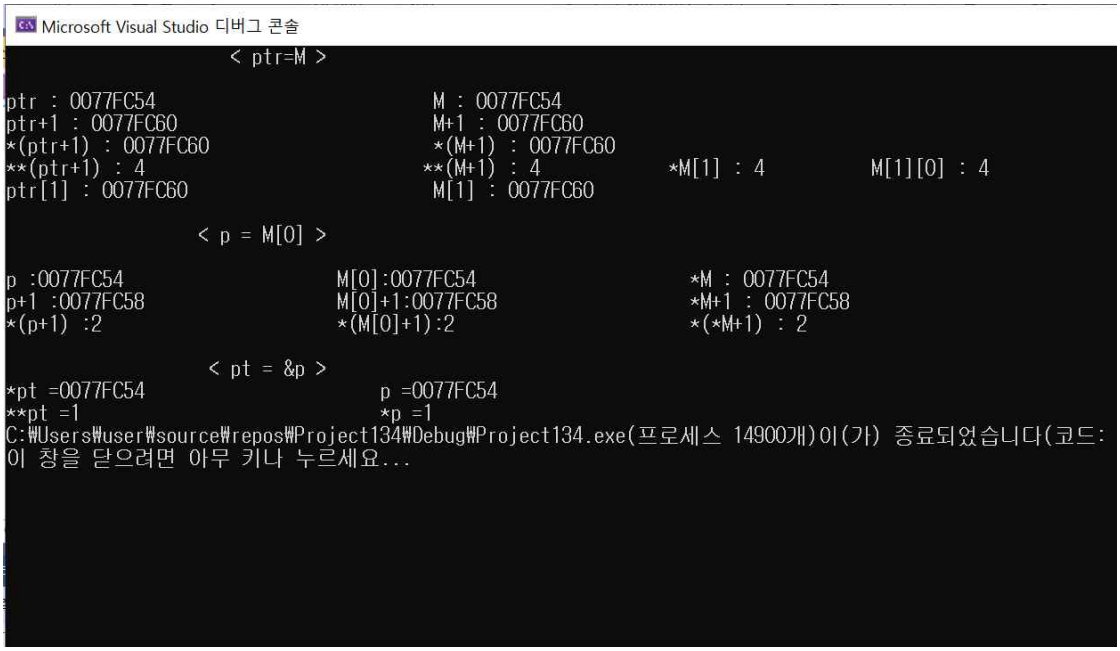
학번 2020039009

이름 차현아

담당교수 최경주 교수님

실습#2-1. 2차원 배열과 포인터(배열 포인터, 이중포인터)

<실행화면>



(요약)

1. 2차원 배열 M 전체를 가리키는 배열포인터 ptr

배열의 이름 = 첫 번째 원소의 주솟값 = 포인터 상수

M= *M = M[0] = &M[0] = &M[0][0] (주소)

M+1 = *(M+1) = M[1] = &M[1] = &M[1][0] (주소)

**M = *M[0] = M[0][0] (데이터)

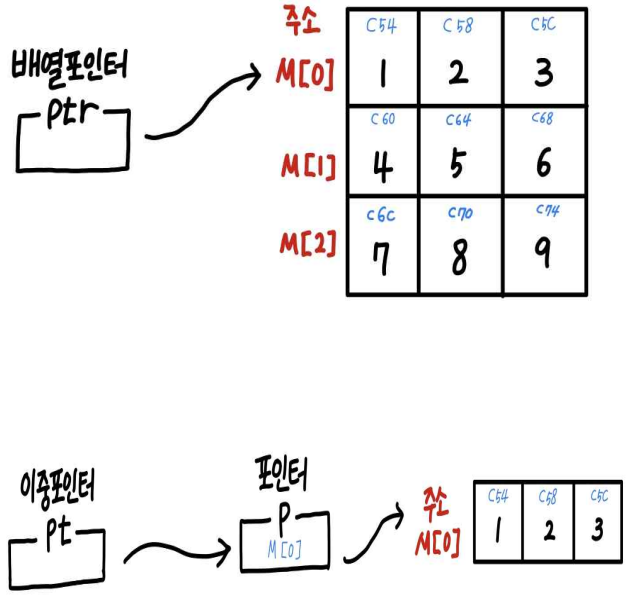
**(M+1) = *M[1] = M[1][0] (데이터)

**(M[1]+2)=M[1][2] (데이터)

2. 2차원 배열의 1행을 가리키는 포인터 p

일반 포인터로는 2차원 배열을 가리킬 수 없다.

2차원 배열을 가리키려면 배열 포인터를 사용해야한다.

이해를 돕기 위한 그림	
	주소값은 16진수로 출력된다. 정수가 입력되는 배열이기 때문에 각 인덱스들의 주소 차이는 4바이트이다. 따라서 M[0][0]은 54(10진수:84),M[1][0]은 (4*3)12가 더해진 60(10진수:96)가 나오게 된다. 처음에 16진수로 출력 된다는 것을 인식하지 못해서 오류가 나는걸로 생각했다. 모든 인덱스의 주소값을 출력해보고나서 16진수로 출력된다는 것이 생각났다.

● 다음 소스코드를 이해하기 위해 꼭 이해하고 있어야하는 점

1. 배열 포인터는 **2차원 배열 전체**를 가리킨다.
2. M[0],M[1],M[2]는 1차원 배열들의 주소값이다. ***중요***
   - M[0]은 주소값이고, M[0][0]은 실제 데이터 값이다.
3. M=M[0](주소), *M=M[0](주소), **M=*M[0]=M[0][0](데이터 값)이다.

소스코드	

```
#include<iostream>
using namespace std;
int main(void)
{
    int M[3][3] = { {1,2,3},{4,5,6},{7,8,9} };

    int(*ptr)[3]; // 배열포인터
    int* p; // 포인터
    int** pt; //이중포인터

    ptr = M; // 배열포인터가 M을 가리키게함.
```

```
ptr = M; // 배열포인터가 M을 가리키게함.

cout.width(30);
cout << "< ptr=M >" << "₩n" << "₩n";
cout << "ptr : " << ptr;// 포인터가 가리키는 곳(=M)의 주소
cout.width(30);
cout << "M : " << M << "₩n"; // 배열의 이름= 첫번째 원소의 주소값
cout << "ptr+1 : " << ptr + 1; // = M+1 == M[1] == &M[1](주소값)
cout.width(30);
cout << "M+1 : " << M + 1 << "₩n"; // == &M[0]+12(4*3) == &M[1] = M[1][0]의 주소값
cout << "*(ptr+1) : " << *(ptr + 1); //== *(M+1) == &M[1][0] (주소값)
cout.width(30);
cout << "*(M+1) : " << *(M + 1) << "₩n"; //M[1](주소값)
cout << "**(ptr+1) : " << **(ptr + 1); // *M[1]== M[1][0] (데이터값)
cout.width(36);
cout << "**(M+1) : " << **(M + 1); // *M[1] == M[1][0](데이터값)
cout.width(20);
cout.width(20);
cout << "*M[1] : " << *M[1]; // 데이터값
cout.width(20);
cout << "M[1][0] : " << M[1][0] << "₩n";
cout << "ptr[1] : " << ptr[1]; // == M[1](주소값)
cout.width(30);
cout << "M[1] : " << M[1] << "₩n" << "₩n"; // = M+1 == M[1] == &M[1](주소값)


p = M[0]; // 포인터가 배열 M의 1행(1차원 배열)을 가리키게함.

cout.width(30);
cout << "< p = M[0] >" << "₩n" << "₩n";
cout << "p :" << p; //== M[0] (주소값)
cout.width(25);
cout << "M[0]:" << M[0];
cout.width(25);
cout << "*M : " << *M << "₩n"; // =M[0](주소값)
cout << "p+1 :" << p + 1; //M[0][1]의 주소
cout.width(25);
cout << "M[0]+1:" << M[0] + 1; // == M[0][1]의 주소 == M[0] + 4바이트
cout.width(25);
cout << "*M+1 : " << *M + 1 << "₩n"; //*M=M[0] -> M[0]+1 == M[0][1]의 주소
cout << "*(p+1) :" << *(p + 1); //*(M[0]+1)== M[0][1]
cout.width(32);
cout << "*(M[0]+1):" << *(M[0] + 1); // ==  M[0][1]
cout.width(32);
cout << "*(*M+1) : " << *(*M + 1) << "₩n" << "₩n"; //*M=M[0] -> *(M[0]+1)


pt = &p;
cout.width(30);
cout << "< pt = &p >" << "₩n";
cout << "*pt =" << *pt; //이중포인터 pt가 가리키는 곳의 값
cout.width(25);
cout << "p =" << p << "₩n"; // 포인터 p의 값
cout << "**pt =" << **pt; //(*pt)=(p)가 가리키는 곳의 값
cout.width(32);
cout << "*p =" << *p; //p가 가리키는 곳의 값
```

배열포인터가 2차원 배열을 가리키게 했다. ptr에는 포인터가 가리키는 곳(M)의 주소가 할당된다. 배열의 이름(M)은 첫 번째 원소의 주소값이며, M+1은 &M[0]+12인 M[1][0]의 주소값이다.
*(M+1)은 &M[1]에 들어있는 값이고,&M[1][0]이다.
**(M+1)은 &M[1][0]에 들어있는 값이고 M[1][0]이다.

포인터 p가 일차원 배열 M[0]을 가리키게 했다.
p에는 가리키는 곳(M)의 주소가 할당되고, M[0]은 &M[0][0]이다. *M은 M[0]이고, &M[0][0]이다. (*M=M[0])
*M+1은 &M[0]+4인 &M[0][1]이다.
*(M[0]+1)은 M[0][1]이다. *M=M[0]이므로 *(*M+1)은 M[0][1]이다.

실습#2-1 소스코드 외의 사항들

› int(*ptr)[3]로 표기하는 이유

- 포인터 ptr의 입장에서는 가리킬 배열이 몇개의 데이터를 가지고 있는지 알 수 없기 때문에 열의 개수를 알려줘야한다. 그래야 포인터가 행을 옮겨 각각의 인덱스들을 가리킬때, (4바이트*열의개수)만큼씩 증가시킨 올바른 주소값을 출력할 수 있다.

요약하자면 [3]으로 ptr이 가리키는 주소가 sizeof(int)*3씩 증가하도록 지정을 해준 것이다. 열의 개수를 지정해주지 않으면 포인터는 배열이 몇개의 인덱스를 가지는지 알 수가 없기 때문에 오류가 발생하게된다.

› int (*ptr)[3] VS int *ptr[3] (괄호의 차이)

- 배열 포인터를 선언할때는 포인터부분에 괄호를 꼭 해주어야한다. 괄호를 해주어야 *연산자가 먼저 적용 되어서 ptr이 먼저 포인터가 되어 int[3]을 가리키는 포인터가 된다.

› (배열포인터)ptr =M 가 정상 작동하는 이유

- ptr은 **2차원 배열 전체**를 가리키는 배열포인터이기 때문이다. 일차원 배열 arr[5]을 만들어 이것을 가리키게 해보았는데 에러메세지가 떴으며, 또한 M[1][0]를 가리키게 해보았는데 이 또한 에러메세지가 출력되었다. 배열 포인터는 다차원 배열의 시작주소를 저장하여 데이터로 사용하는 포인터 변수임을 알 수 있었다.

› (포인터)p=(2차원 배열)M 일 때 오류가 발생하는 이유

M은 2차원 배열이기 때문에 에러가 나게 된다. 일반 포인터로는 2차원 배열을 가리킬 수 없다. 2차원 배열을 가리키려면 배열 포인터를 사용해야한다.

+ 포인터 p에서 M[1][0](1행 외의 행)에 접근하려면 어떻게 해야하는가?

- 마찬가지로 일반 포인터로 2차원 배열을 가리킬 수 없다.

› *M+1은 *(M+1)와 같은가?

*M=M[0], *M+1=M[0]+1=M[0][1]의 주소값이고, *(M+1)=M[1]=M[1][0]의 주소값이다. 괄호가 있는 *(M+1)은 더하기 연산이 먼저 일어나서 M[1]에 *이 붙는 것이고, *M+1은 *M에 1이 더해지는 것이다.

실습#2-2. const 포인터

<CASE A>

```
int main(void)
{
    int i1 = 10;
    int i2 = 20;
    const int* pInt1; // 포인터가 가리키는 변수에 const 속성 부여
                      // 포인터가 가리키고 있는 변수의 값 수정불가
    pInt1 = &i1;
    *pInt1 = 30;  // 포인터가 가리키고 있는 변수의 값을 수정하는 것이기 때문에 수정 불가
    i1 = 300;     //포인터 변수를 이용하지 않고 변수에 직접 접근 했기때문에 수정 가능
                  // 만약 const in i1= 을 했더라면 수정 불가
}
```

const 속성이란 변수의 값을 수정할 수 없게 하는 것이다. CASE.A 에서는 포인터가 가리키는 변수에 const 속성을 부여했기 때문에 *pInt1의 값은 수정 할 수 없다. 하지만 같은 영역인 i1의 값은 수정이 가능하다. 그 이유는 접근방식의 차이인데, i1의 값을 바꿀 때는 포인터를 거치지지 않고 직접적으로 접근 했기 때문이다. 만약 i1을 선언할 때 const 속성을 부여하였으면, 마찬가지로 i1의 값도 바꿀 수 없게 된다.

<CASE B>

```
int main(void)
{
    int i1 = 10;
    int i2 = 20;
    int* const pInt2=&i1; // 포인터  변수 자체에 const 속성 적용

    pInt2 = &i2;        // 포인터 변수 자체 값 수정 불가
    *pInt2 = 50;        // 포인터가 가리키고 있는 변수의 값은 수정 가능
}
```

포인터 변수 자체에 const 속성이 적용되었다. 따라서 포인터 변수 자체의 값을 수정할 수 없다. 하지만 포인터가 가리키고 있는 변수의 값은 수정이 가능하다.

<CASE C>

```
int main(void)
{
    int i1 = 10;
    int i2 = 20;
    const int* const p=&i2;    // 포인터 변수 자체와 포인터가 가리키는 변수에 const 속성 적용

    p= &i1;                    // 포인터 변수 자체 값 수정 불가
    *p = 40;                   // 포인터가 가리키고 있는 변수 값 수정 불가
}
```

포인터 변수와 포인터가 가리키는 변수에 const 속성이 적용된 경우이다. 위 경우에서는, 포인터 변수의 자체 값과 포인터가 가리키는 변수의 값은 수정할 수 없게 된다. 하지만 i1과 i2의 값은 수정이 가능하다. 모든 값들을 변경이 불가능하게 하려면 i1과 i2를 선언할 때 int 앞에 const를 써주면 된다.