C++과 계산

○ 고려대학교 수학과

2011년 5월 31일

제 1 장

C++언어

제 1 절 C++언어에서의 계산

1.1 포인터

포인터는 메모리의 주소를 가리키는 변수이다. 사용자에 의해 어떤 변수가 선언되면 그 변수의 주소는 컴파일러에 의해 자동으로 지정된다. 또한 선언된 변수의 주소에 대하여 포인터를 사용하여 변수로써 사용할 수 있다. C++언어를 사용하는데 있어서 변수 주소를 반드시 알아야 하는 것은 아니다. 하지만 그 주소를 포인터로 지정하여 쓰면 변수 이름을 통해 데이터와 메모리를 효율적으로 쓸 수 있는 이점이 있다. 만일 변수들을 지역 변수로 쓴다면 나중에 함수가 반환될 때 지역 변수들은 해제되어 더 이상 변수를 사용할 수 없게 된다. 또한 전역 변수로 선언되면 모든 부분에서 제한 없이 접근할 수 있으나 가독성이 떨어지고 코드의 유지 보수가 어려운 단점이 있다.

그림 1.1: 포인터에서 위 3 가지를 혼동하지 말자.

```
#include<iostream>
typedef unsigned short int USHORT;
using namespace std;
int main()
{
USHORT myAge;
USHORT *pAge=0;
myAge=5;
cout<<"myage: "<<myAge <<"\n";
pAge=&myAge;
cout<<"*pAge:"<<*pAge<<"\n\n";

cout<<"Let *pAge=7 \n";
*pAge=7;
cout<<"*pAge: "<<*pAge<<"\n";
cout<<"myAge: "<<myAge<<"\n\n";

cout<<"Let myAge=9  \n";
myAge=9;
cout<<"myAge: "<<myAge<<"\n";
cout<< "*pAge: "<<*pAge<<"\n";

 return 0;
}
```

위 코드에서는 첫 번째 myAge에 5의 값을 대입하고 그 주소를 pAge에 전달하여 myAge값을 출력하였다. 두 번째는 포인터로 myAge값을 7로 바꾸어 주고 출력하였다. 마지막에는 myAge에 바로 값을 입력하고 출력하였다. 주의할 점은 포인터를 선언할 때 *를 사용하고 pAge에 myAge의 주

소를 대입할 때에 &을 사용한다.

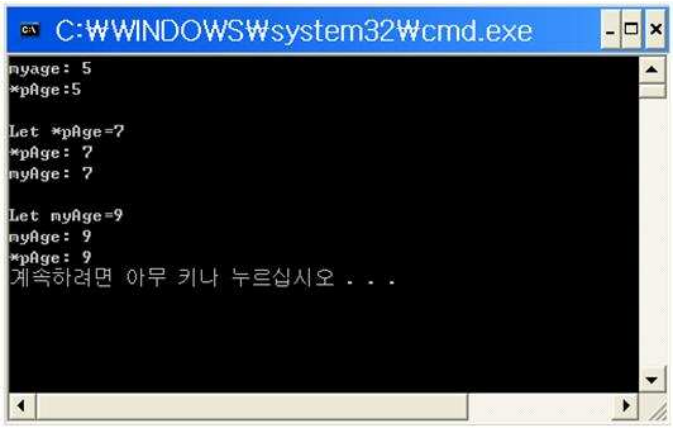

그림 1.2: 포인터를 사용한 연산.

1.2 new와 delete

자유 기억 공간을 가리키는 포인터에 메모리 할당(new)과 제거(delete)를 알아 보자. new라는 예약어를 통해 자유 기억 공간에 메모리를 할당할 수 있다. 그리고 메모리 가지고 작업을 다하였으면 그 포인터를 delete를 사용하여 반환하여야 한다. 왜냐하면 new를 통해 할당된 메모리는 자동으로 해제되지 않기 때문에 delete를 하여 메모리 누수를 막아 주어야 한다..

```
#include<iostream>
typedef unsigned short int USHORT;
using namespace std;
int main()
{
int localVariable=5;
int *pLocal=&localVariable;
int *pHeap=new int;
```

```
*pHeap=7;
cout<<"localVarible: "<<localVariable<<"\n";
cout<<"*pLocal:"<<*pLocal<<"\n";
cout<<"*pHeap:"<<*pHeap<<"\n";
delete pHeap;
pHeap=new int;
*pHeap=9;
cout<<"*pHeap: "<<*pHeap<<"\n";
delete pHeap;
 return 0;
}
```

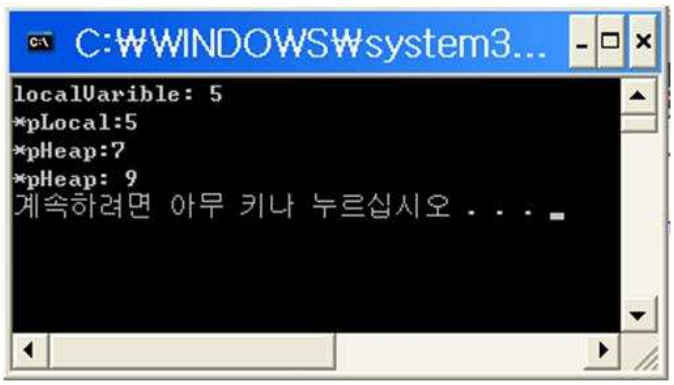

그림 1.3: 포인터를 사용한 연산.

1.3 function

프로그램 안에서 함수(subprogram)를 사용하려면 먼저 메인(main)함수 앞에 사용하고자 하는 함수를 정의해 주어야 한다. 선언할 때에는 컴파일러에게 함수의 이름, 반환값, 그 함수의 매개변수들을 알려준다. 그리고 메인함수를 읽게 된다. 메인 함수에서 컴파일러는 순차적으로 코드를 읽게 된다. 만일 메인함수 안에서 부함수(subprogram)을 만나면 그 함수로 넘어가

서 실행되고 함수가 끝나면 본래 순서(메인함수에서의 다음 줄)로 돌아 온다. 유의할 점은 부함수에서 할당된 지역 변수는 메인 함수로 돌아왔을 때 더이상 유효하지 않는 것을 기억하자.

```
#include<iostream>

int Add(int x, int y)
{
std::cout << "In Add(), recived " << x << " and " << y <<
"\n";
    return (x+y);
}
int main()
{
using namespace std;
cout<< "I am in main function()\n";
int a,b,c;
cout<<"Give me two integers";
cin>>a;
cin>>b;
cout<<"\n Now, I am calling add()\n";
c=Add(a,b);
cout<<"\n I am back in main()\n";
cout<<"sum of two integers is "<<c;
cout<<"\n";

return 0;
}
```

그림 1.4: class method.

1.4 class

클래스의 구조는 자료와 함수를 동시에 가진 캡슐 모양을 가지고 있습니다. 즉, 정수형이나 문자형과 함수로 새로운 형태의 자료형을 만드는 것이 클래스(class)이다. 클래스 선언할 때 전용(private)과 범용(public)이라는 예약어가 사용된다. 전용은 그 클래스안의 방법(method)에 의해서만 접근된다. 따라서 그 클래스 범위 내에서만 한정된다. 반면 범용은 다른 클래스 객체에 의해서도 접근될 수 있다.

```
#include<iostream>

using namespace std;

class Cat
{
public:
int itsAge;
int itsWeight;
};

int main()
{
```

```
Cat Franky;
Franky.itsAge=5;
cout<<"Franky is a cat who is";
cout<< Franky.itsAge<< "years old.\n";

 return 0;
}
```

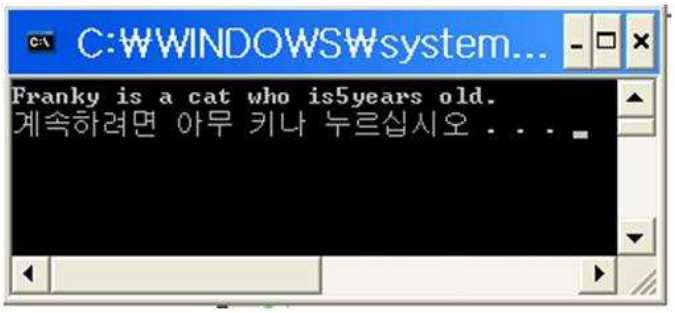

그림 1.5: class의 이용.

접근자 방법(accessor method)
우회적인 접근 방법을 선택하여 프로그램을 안정적으로 실행할 수 있다.

```
#include<iostream>

using namespace std;

class Cat//클래스 정의 시작
{
public://범용 함수
int GetAge();//접근자 함수
void SetAge(int age);//접근자 함수
```

```
void Meow();

private://전용 함수
int itsAge;
};

int Cat::GetAge()
{
return itsAge;
}
void Cat::SetAge(int age)
{
//member variable인 itsAge에
//age로 전달된 값 설정하기/
itsAge=age;
}
//method 정의 매개 변수는 없음.
void Cat::Meow()
{
cout<<"meow.\n";
}
int main()
{
Cat Franky;
Franky.SetAge(5);
Franky.Meow();
cout<<"Franky is a cat who is";
cout<< Franky.GetAge()<< "years old. really?\n";
Franky.Meow();
```

```
 return 0;
}
```

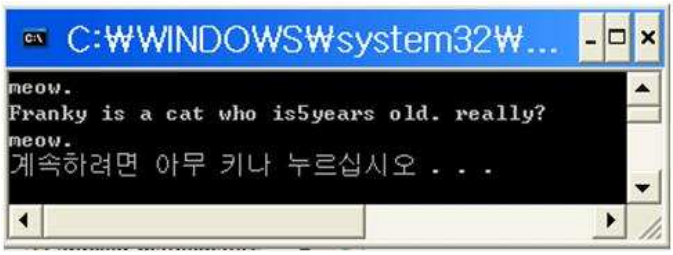

그림 1.6: class method.

1.5 상속과 파생(Inheritance and Derivation)

더 모호하거나 큰 개념의 부모 클래스에 있는 멤버를 자식 클래스가 그대로 받아 새롭게 정의하거나 수정할 수 있다. 기존의 클래스에서 새로운 기능을 추가한 새로운 클래스를 원래 클래스에서 파생되었다고 한다.

```
#include<iostream>

using namespace std;

enum BREED {MOUNGMOUNG_YI,RURU};

class Mammal//부모 클래스
{
public://범용 함수
//생성자
Mammal():itsAge(2),itsWeight(5){};
```

```
//소멸자
~Mammal(){};

int GetAge() const {return itsAge;}//접근자 함수
void SetAge(int age){itsAge=age;}//접근자 함수
int GetWeight() const{return itsWeight;}
void SetWeight(int weight){itsWeight=weight;}

//methods
void Speak() const{cout<<"mammal sound-\n";}
void Sleep() const{cout<<"shh. I'm gonna sleep.\n";}

protected:  //상속해 주기 위해선
//private가 아니라 protected를
//쓴다.
int itsAge;
int itsWeight;
};

class Dog : public Mammal//부모 클래스를 써준다.
{
public:
//생성자
Dog():itsBreed(RURU){}
~Dog(){};

//접근자
BREED GetBreed() const {return itsBreed;}
void SetBreed(BREED breed) {itsBreed=breed;}
```

```
void WagTail() const {cout<<"Tail wagging..\n";}
void BegForFood() const {cout<<"Begging for food  \n";}
private:
BREED itsBreed;
};


int main()
{
Dog Pole;
Pole.Speak();
Pole.WagTail();
cout<<"Pole is "<<Pole.GetAge()<<" years old\n";

 return 0;
}
```

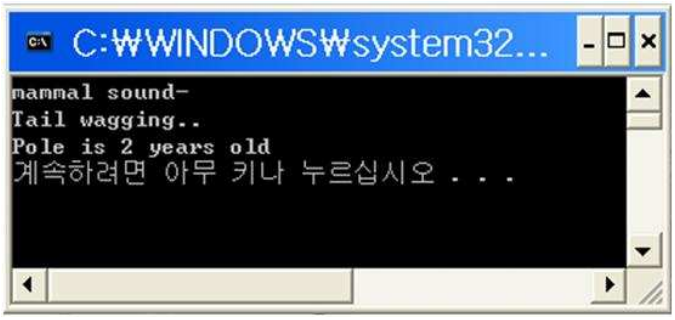

그림 1.7: class method.

Virtual Method

C++는 기본 클래스를 가리키는 포인터에 파생 클래스 객체를 대입하

는 다형성을 제공한다. 따라서 부모 클래스에 포인터를 넣는 것이 가능하게 한다.

```
#include<iostream>
using namespace std;



class Mammal//부모 클래스
{
public://범용함수

Mammal():itsAge(1){cout<<"Mammal constructor.\n";}
virtual ~Mammal(){cout<<"Mammal destructor.\n";}
void Move() const{cout<<"All mammals walk crawl or fly\n";}
virtual void Speak() const{cout<<"Mammal speak? or bark?
\n";}

protected:  //상속해주기위해선
//private가 아니라 protected를
//쓴다.
int itsAge;
};

class Dog : public Mammal//부모 클래스를 써준다.
{
public:
//생성자
Dog(){cout<<"Dog Constructor\n";}
virtual ~Dog(){cout<<"Dog destructor\n";}
```

```
void WagTail() const {cout<<"Tail wagging..\n";}
void Speak() const {cout<<"Dog Speaks English: Bow-wow\n";}
void Move() const {cout<<"Dog jumps\n";}
};


int main()
{
Mammal *pDog=new Dog;
pDog->Move();
pDog->Speak();

 return 0;
}
```

여기서 호출한 함수는 Mammal 포인터를 가지고 있는 것을 알고 있지만 Dog의 method가 호출된다.

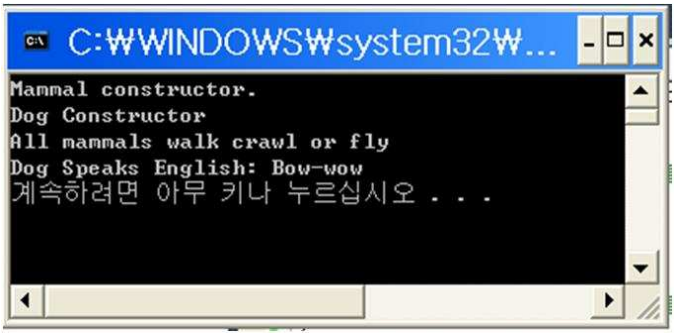

그림 1.8: template을 사용한 max값 구하기.

이제 virtual function을 이용하여 보자.

1.6 템플릿(Template)

템플릿을 이용하면 int나 float, 사용자가 만든 클래스에 상관없이 사용할 수 있다. 이번에는 템플릿을 이용한 함수를 살펴보자.

```
#include<iostream>
#include<fstream>
using namespace std;



template<typename T>
T max1(T a, T b)
{
return (a>b ? a:b);
}



int main()
{ int int1=5,int2=11;
  int int3=max1(int1,int2);
  double double1=1.1,double2=17.17;
  double double3=max1(double1,double2);

cout<<"The sum of two integers is "<<int3;
cout<<"\n";
cout<<"The sum of two rational numbers is "<<double3;
cout<<"\n";

return 0;
}
```

위 함수는 모든 타입에 사용할 수 있는 max함수이다. 임의의 타입인 T을 template < typename T > 을 사용하여 선언하였다.

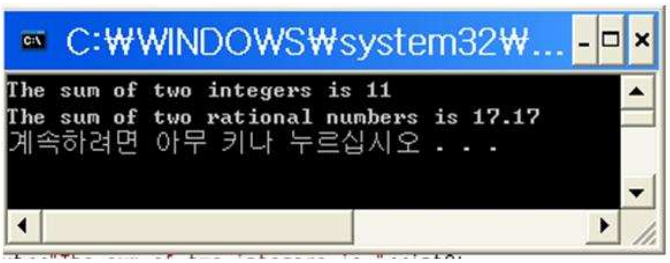

그림 1.9: template을 사용한 max값 구하기.

1.7 벡터 계산

$\mathbb{R}^n$에서 벡터 u, v가 있다고 할 때, 이 두 벡터의 내적(inner product :=$u^T v$)을 구해 보자.

$$\begin{aligned} \mathbf{u} &= [u_1 u_2 \cdots u_n]^T \text{ and } \mathbf{v} = [v_1 v_2 \cdots v_n]^T \\ \mathbf{u}^T \cdot \mathbf{v} &= u_1 v_1 + u_2 v_2 + u_3 v_3 + \cdots u_n v_n. \end{aligned}$$

C++에서 내적을 구하기 위해 변수형(int, double, float, etc)을 정해준다. 그리고 각 벡터의 index는 0부터 시작하게 된다. 따라서 벡터 array **u**와 **v**의 내적은 각 index에 -1을 한 값으로 계산된다. 즉,

$$\mathbf{u}^T \cdot \mathbf{v} = u[0] * v[0] + u[1] * v[1] + u[2] * v[2] + \cdots u[n-1] * v[n-1].$$

```
#include <iostream>
using namespace std;
#define N 3
int main(){
int i;
int *u, *v, w;
u=new int [N];
```

```
v=new int [N];
u[0]=4; u[1]=5; u[2]=-1;
cout << endl << "Vector u:" << endl;
for (i=0;i<N;i++)
cout << u[i] << " ";
cout << endl;
v[0]=-2; v[1]=3; v[2]=7;
cout << "Vector v:" << endl;
for (i=0;i<N;i++)
cout << v[i] << " ";
cout << endl;
w=0;
for (i=0;i<N;i++)
w += u[i]*v[i];
cout << "the product w=u.v is " << w << endl;
delete u,v;
return 0;}
```

내적을 구한 결과값이다.

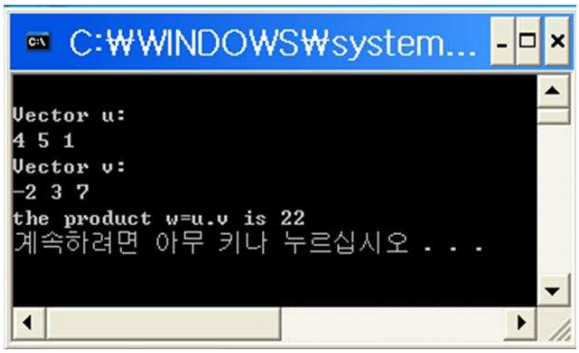

그림 1.10: 내적하기.

1.8 행렬 계산

행렬 $m \times n$ A와 $n \times m$ B가 있다고 하자. 그러면 그 곱의 행렬 $m \times m$ C는 원소 a_{ij}와 $b_{i',j'}$의 곱과 합으로 표시 된다.

$$\begin{aligned} c_{11} &= a_{11}b11 + a_{12}b_{21} + \cdots a_{1n}b_{n1} \\ &\text{간단히 쓰면} \\ c_{11} &= \sum_{k=1}^{n} a_{1k}b_{k1}. \end{aligned}$$

그러면 행렬 C와 그 요소 c_{ij}, $(1 \le i \le m,\ 1 \le j \le n)$의 요소는

$$c_{ij} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{pmatrix} = \begin{pmatrix} \sum_{k=1}^{n} a_{1k}b_{k1} & \sum_{k=1}^{n} a_{1k}b_{k2} & \cdots & \sum_{k=1}^{n} a_{1k}b_{kn} \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{k=1}^{n} a_{mk}b_{k1} & \sum_{k=1}^{n} a_{mk}b_{k2} & \cdots & \sum_{k=1}^{n} a_{mk}b_{kn} \end{pmatrix}$$

이다.
예를 들어, $m \times p$ 행렬 A의 원소$a(i,j) = (i-1) + (j-1)$와 $p \times n$ 행렬 B의 원소 $b(i,j) = (i-1) - (j-1)$의 곱 AB를 구해 보자. 그러면

$$A = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \end{pmatrix}, B = \begin{pmatrix} 0 & -1 \\ 1 & 0 \\ 2 & 1 \\ 3 & 2 \end{pmatrix} \text{ 이고 } C = \begin{pmatrix} 14 & 8 \\ 20 & 10 \\ 26 & 12 \end{pmatrix} \text{ 가 된다.}$$

C++로 확인해 보자.

```
#include <iostream>
#define M 3
#define P 4
#define N 2
```

```
using namespace std;
int main()
{
int i,j,k;
int **a,**b,**c;
a = new int *[M];
b = new int *[P];
c = new int *[M];
for (i=0;i<M;i++)
a[i]=new int [P];
for (j=0;j<P;j++)
b[j]=new int [N];
for (k=0;k<=M;k++)
c[k]=new int [N];

for (i=0;i<M;i++){
for (j=0;j<P;j++){
a[i][j]=i+j;
}
}
for (i=0;i<P;i++){
for (j=0;j<N;j++){
b[i][j]=i-j;
}
}

cout << "Matrix A:" << endl;
for (i=0; i<M; i++)
{
```

```
for (j=0;j<P;j++)
cout << a[i][j] << " ";
cout << endl;
}

cout << endl << "Matrix B:" << endl;
for (i=0; i<P; i++)
{
for (j=0; j<N; j++)
cout << b[i][j] << " ";
cout << endl;
}
cout << endl << "Matrix C (A multiplied by B):" << endl;
for (i=0; i<M; i++)
{
for (j=0;j<N;j++)
{
c[i][j]=0;
for (k=0;k<P;k++)
c[i][j] += a[i][k]*b[k][j];
cout << c[i][j] << " ";
}
cout << endl;
}
for (i=0;i<M;i++)
delete a[i];
for (j=0;j<P;j++)
delete b[j];
for (k=0;k<M;k++)
```

```
delete c[k];
delete a,b,c;
return 0;
}
```

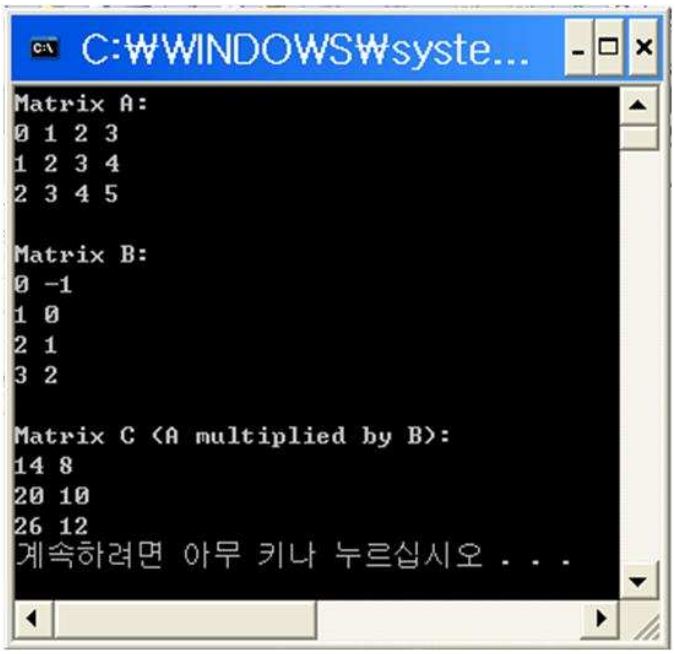

그림 1.11: 행렬 곱하기

여러가지 행렬 연산

이제 역행렬과 행렬의 더하기 빼기 등, 여러가지 행렬 연산을 해보자. 다음과 같은 행렬 연산이 있다고 하고 그 계산된 값을 Z라 한다.

$$Z = A^2B^{-1} + A^{-1}B - AB$$

계산은 아래의 그림 순서대로 하자. C++코드는 다음과 같다.

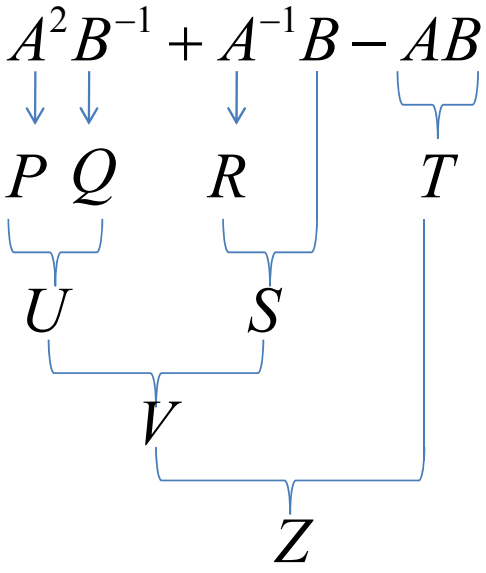

그림 1.12: 행렬 곱하기

```
#include <fstream>
#include <iostream>
using namespace std;
#define N 3

class MatAlgebra
{
public:
MatAlgebra() { }
~MatAlgebra() { }
void ReadData(double **,double **);
void MatAdd(bool,double **,double **,double **);
```

```
void MatInverse(double **,double **);
void MatMultiply(double **,double **,double **);
};

void MatAlgebra::ReadData(double **a,double **b)
{
int i,j;
    ifstream InFile("Code2F.in");
    for (i=1;i<=N;i++)
        for (j=1;j<=N;j++)
            InFile >> a[i][j];
    for (i=1;i<=N;i++)
        for (j=1;j<=N;j++)
            InFile >> b[i][j];
    InFile.close();
}

void MatAlgebra::MatAdd(bool flag,double **c,double
**a,double **b)
{
int i,j,k;
for (i=1;i<=N;i++)
for (j=1;j<=N;j++)
c[i][j]=((flag)?a[i][j]+b[i][j]:a[i][j]-b[i][j]);
}

void MatAlgebra::MatMultiply(double **c,double **a,double
**b)
{
```

```
int i,j,k;
for (i=1;i<=N;i++)
for (j=1;j<=N;j++)
{
c[i][j]=0;
for (k=1;k<=N;k++)
     c[i][j] += a[i][k]*b[k][j];
}
}

void MatAlgebra::MatInverse(double **x,double **a)
{
    int i,j,k;
    double Sum,m;
double **b,**q;
b=new double *[N+1];
q=new double *[N+1];
for (i=0;i<=N;i++)
{
b[i]=new double [N+1];
q[i]=new double [N+1];
}
for (i=1;i<=N;i++)
     for (j=1;j<=N;j++)
{
b[i][j]=0;
q[i][j]=a[i][j];
if (i==j)
b[i][j]=1;
```

```
}

// Perform row operations
     for (k=1;k<=N-1;k++)
        for (i=k+1;i<=N;i++)
{
m=q[i][k]/q[k][k];
for (j=1;j<=N;j++)
{
q[i][j]-=m*q[k][j];
b[i][j]-=m*b[k][j];
}
}

// Perform back substitution
    for (i=N;i>=1;i--)
for (j=1;j<=N;j++)
{
Sum=0;
x[i][j]=0;
for (k=i+1;k<=N;k++)
Sum += q[i][k]*x[k][j];
x[i][j]=(b[i][j]-Sum)/q[i][i];
}
for (i=0;i<=N;i++)
delete b[i],q[i];
delete b,q;
}
```

```
int main()
{
int i,j;
double **A,**B;
double **P,**Q,**R,**S,**T,**U,**V,**Z;
MatAlgebra g;

A=new double *[N+1];
B=new double *[N+1];
P=new double *[N+1];
Q=new double *[N+1];
R=new double *[N+1];
S=new double *[N+1];
T=new double *[N+1];
U=new double *[N+1];
V=new double *[N+1];
Z=new double *[N+1];
for (i=0;i<=N;i++)
{
A[i]=new double [N+1];
B[i]=new double [N+1];
P[i]=new double [N+1];
Q[i]=new double [N+1];
R[i]=new double [N+1];
S[i]=new double [N+1];
T[i]=new double [N+1];
U[i]=new double [N+1];
V[i]=new double [N+1];
Z[i]=new double [N+1];
```

```
}
cout.setf(ios::fixed);
cout.precision(12);
g.ReadData(A,B);
g.MatMultiply(P,A,A);
g.MatInverse(Q,B);
g.MatInverse(R,A);
g.MatMultiply(U,P,Q);
g.MatMultiply(S,R,B);
g.MatMultiply(T,A,B);
g.MatAdd(1,V,U,S);
g.MatAdd(0,Z,V,T);
for (i=1;i<=N;i++)
{
for (j=1;j<=N;j++)
cout << Z[i][j] << " ";
cout << endl;
}
for (i=0;i<=N;i++)
delete A[i],B[i],P[i],Q[i],R[i],S[i],T[i],U[i],V[i],Z[i];
delete A,B,P,Q,R,S,T,U,V,Z;
return 0;
}
```

소스 파일Code2F.in은 띄어쓰기와 상관없이 쓴다.

```
5 -2 3
4 7 -1
3 4 8
```

```
-4 8 1
6 -2 7
4 2 5
```

참고 문헌

[1] 이식, 이홍석, 김정한, 이승우, *MPI 병렬 프로그래밍*, Oxford 어드북스(한솜), (2010)

[2] 김민장, *프로그래머가 몰랐던 멀티코어 CPU 이야기*, 한빛미디어, (2010)

[3] 저) 제스 리버티, 역) 박춘, *초보자를 위한 C++*, 정보문화사 (2006)

[4] 이현창, *(뇌를 자극하는)C++ 프로그래밍*, 한빛미디어, (2006)

This work has been achieved by helpful advice from scientific computational lap colleagues.