配列とポインタ

配列を表す変数は、その配列がとられているメモリー領域の先頭を指すポインタになっている。したがって、配列

double　a[10];

において、a をポインタとして扱うと

*(a+1)

は

a[1]

を表す。

配列を表す変数をポインタとして用いたプログラム例をリスト１に示す。

リスト１　配列とポインタ。

```
int main(array<System::String ^> ^args)
{
	double a[3][3];
	for (int i = 0; i < 3; i++)
		for (int j = 0; j < 3; j++)
			a[i][j] = j + i * 3;
	Console::WriteLine("a = ");
	for (int i = 0; i < 3; i++){
		for (int j = 0; j < 3; j++)
			Console::Write((*(*(a+i) + j)).ToString()->PadLeft(3));
		Console::WriteLine();
	}

	double ** b;
	b = new double*[3];
	for (int i = 0; i < 3; i++)
		b[i] = new double[3];
	for (int i = 0; i < 3; i++)
		for (int j = 0; j < 3; j++)
			*(*(b+i) + j) = j + i * 3;
	Console::WriteLine();
	Console::WriteLine("b =");
	for (int i = 0; i < 3; i++){
		for (int j = 0; j < 3; j++)
			Console::Write(b[i][j].ToString()->PadLeft(3));
		Console::WriteLine();
	}

	if (b != nullptr){
		for (int i = 0; i < 3; i++)
			if (b[i] != nullptr) delete [] b[i];
		delete [] b;
	}
```

```
        Console::WriteLine();
        Console::WriteLine("Enterキーを押して終了。");
        Console::ReadLine();
        return 0;
}
```

リスト１を実行すると図１のようになる。

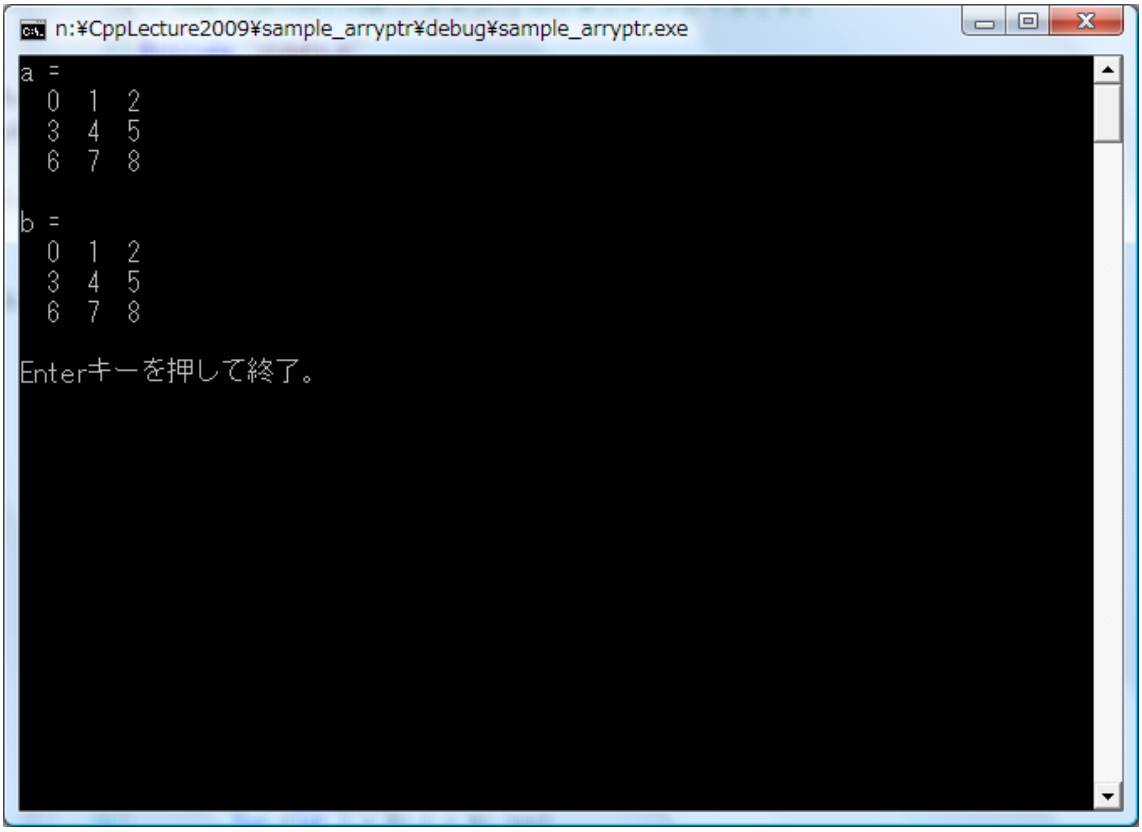

図１　リスト１の実行結果。

リスト１において、最後の方にある delete[]演算子による領域の解放は行わなくてもよいものである。プログラムの実行が終了すると new 演算子で確保された領域は解放される。これを確認するプログラムがリスト２である。

リスト２　配列領域の確保と解放。

```
int main(array<System::String ^> ^args)
{
        double ** a;
        Console::WriteLine("Enterキーを押して進む。");
        Console::ReadLine();
```

```
        a = new double*[10000];
        for (int i = 0; i < 10000; i++)
                a[i] = new double[10000];
        a[0][0] = 1.0;

        Console::WriteLine("Enterキーを押して終了。");
        Console::ReadLine();
        return 0;
}
```

リスト２のプログラムを実行すると図２の画面が表示される。

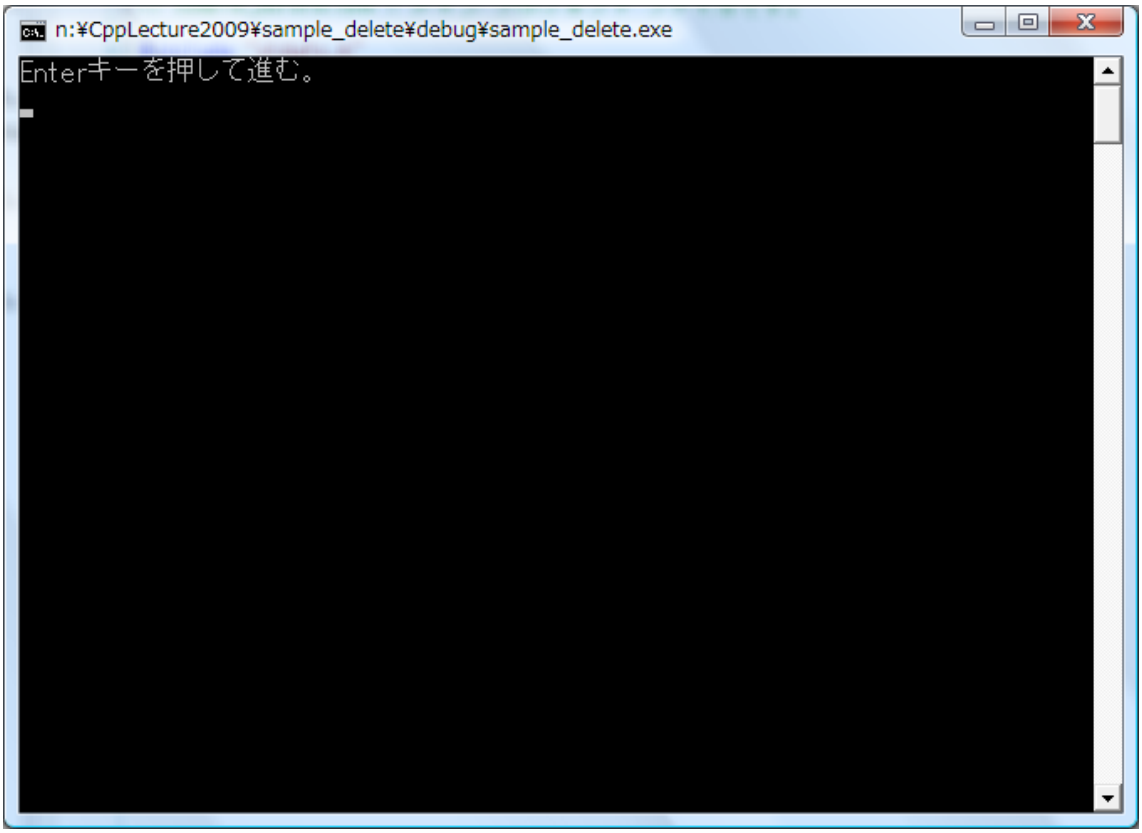

図２　リスト２の実行直後の画面。

図２のときは、まだ配列 a の領域は確保されていない。このときのタスクマネジャーの表示は図３のようになっている。使用メモリの大きさは 1.68GB となっている。

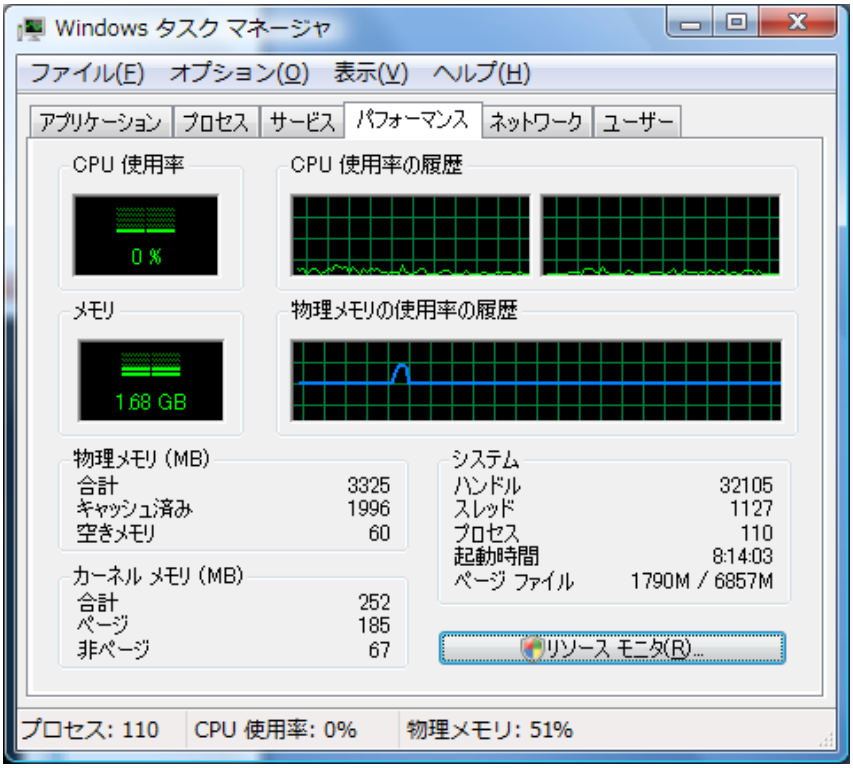

図３　図２のときのタスクマネージャ。

図２の状態で Enter キーを押すと図４のようになる。

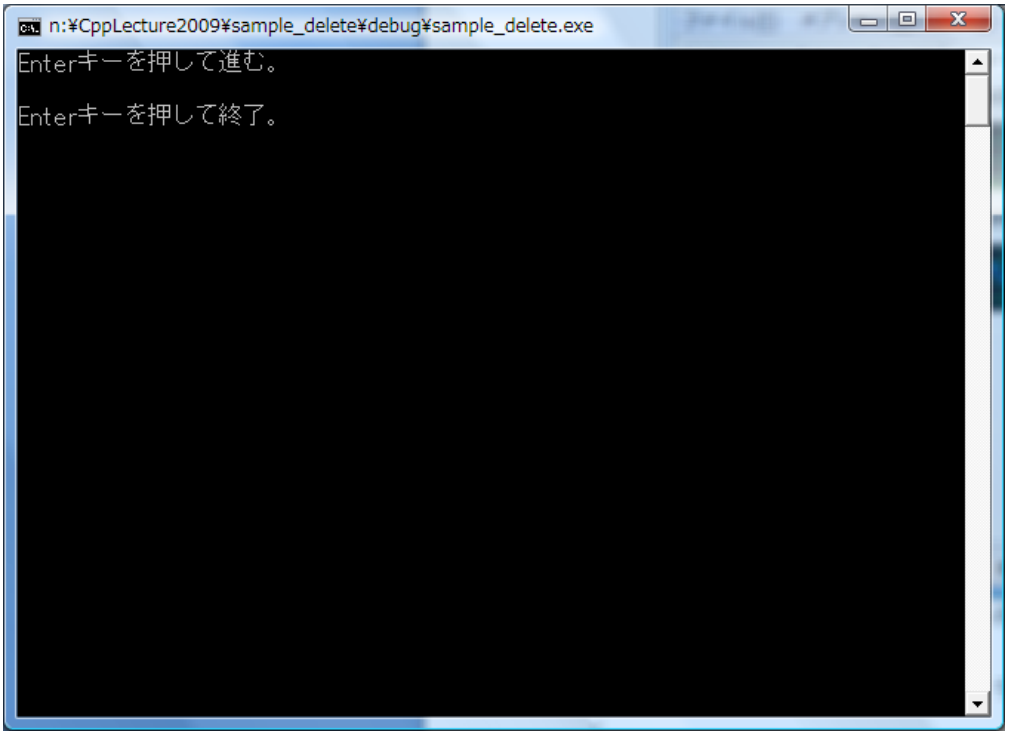

図４　配列 a の領域が確保されている状態。

図４のときは、リスト２からわかるように、配列 a の領域が確保されている状態である。このときのタスクマネージャは図５のようになっている。

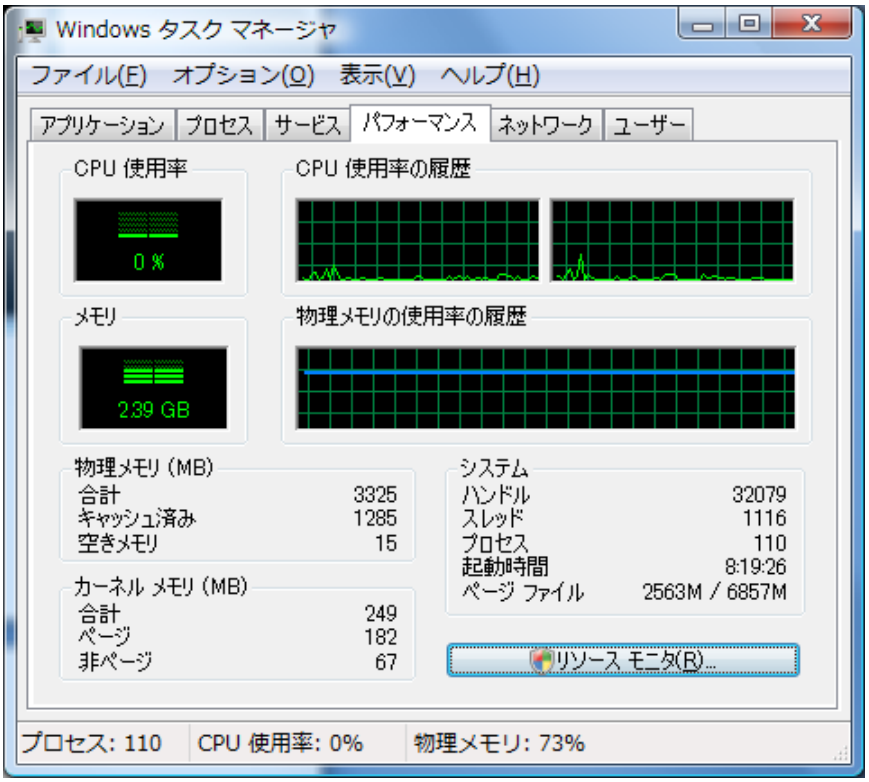

図５　配列の領域が確保されている状態。

図３のタスクマネージャにおけるメモリ使用量よりも値が増えていることがわかる。このメモリ使用量は Windows での他のリソース使用の要因の影響も受けるので、メモリ使用量の増加量が確保した配列の領域の大きさに正確に対応するわけではない。

図４の状態で Enter キーを押すとプログラムの実行終了となり、コンソール画面は消える。プログラム終了後のタスクマネージャは図６のようになっている。メモリ使用量の値の減少から、配列の領域が解放されたことがわかる。リスト２では delete [ ]演算子による解放は行っていない。図５に見られる配列の領域の解放は、プログラムの実行終了に伴って自動的に行われたものである。

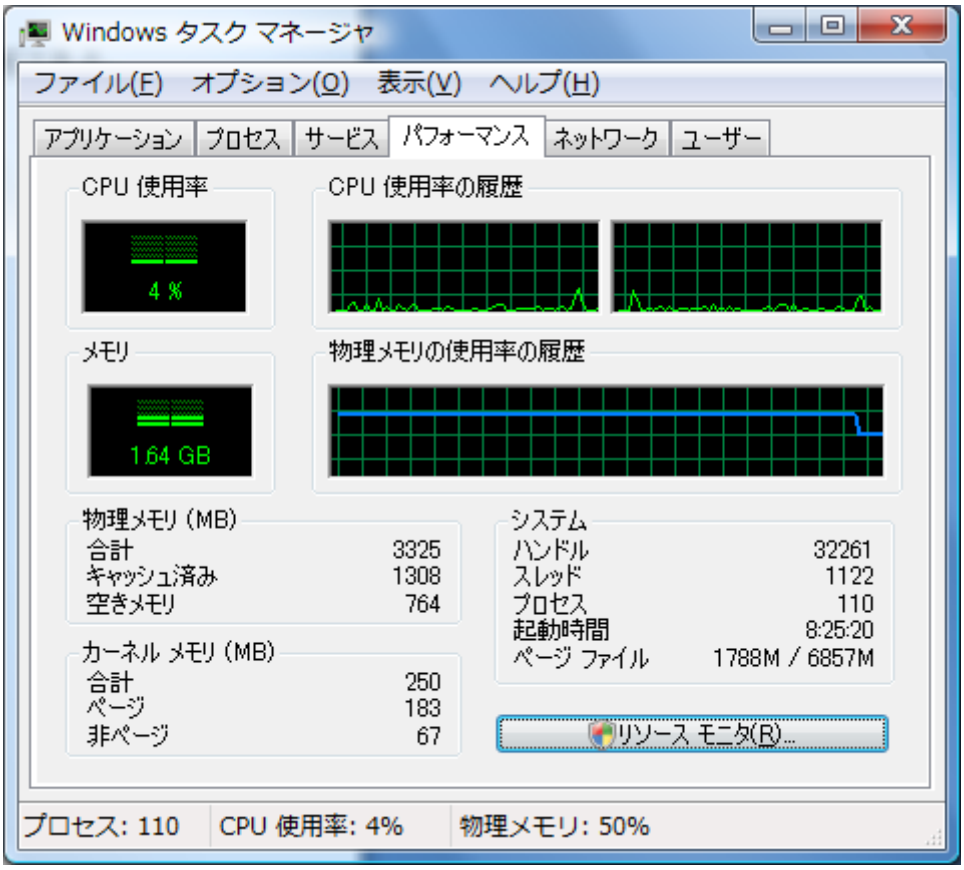

図６　プログラム実行終了後の表示値。