プログラミング及び演習
第4回 演算子/配列1/演習
(2017/05/12)

講義担当
大学院情報学研究科知能システム学専攻
教授　森健策

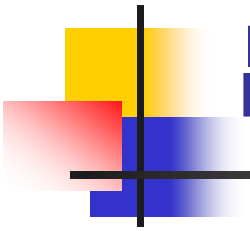
演習担当
大学院情報学研究科知能システム学専攻
小田昌宏

本日の講義・演習の内容

- さわやかな季節です.
  - プログラミングに絶好の季節です.
- 演算子
- 配列1
- 演習
- 講義・演習ホームページ
  - http://www.newves.org/~mori/16Programming

メモを取りましょう

- 講義で補足された点のメモをとる習慣をつけてください。
- 重要なところを赤ペン・イエローマーカー等でマークするのも有効です。
- 折角の講義資料です。いろいろと書き込んでください。



出席

- NUCTにて出席アンケートに回答
  - http://ct.nagoya-u.ac.jp
  - 回答1には今日の合言葉
  - 回答2には連休の感想を一言
  - 回答3には
    「前回の演習課題で一言」を

演算子の一覧

- 教科書 p. 82 表8-1
- 優先順位
  - 値が小さいほど高い
- 結合規則
  - 同じ優先順位の演算子が複数続く場合の結合方向
- 丸括弧は演算子ではないが、優先的に評価する部分を示す

演算子の一覧

優先順位	演算子	結合規則	名称
1	( )	左→右	関数呼び出し演算子
	[ ]	左→右	配列添字演算子
	. ->	左→右	直接メンバ演算子・間接メンバ演算子 （構造体）
	++ --	左→右	後置増分・減算演算子

演算子の一覧

優先順位	演算子	結合規則	名称
2	++ --	左←右	前置増分･減算演算子
	sizeof	左←右	記憶量演算子
	& *	左←右	アドレス演算子･間接参照演算子
	+ -	左←右	正･負符号演算子
	~	左←右	ビット否定演算子
	!	左←右	論理否定演算子

演算子の一覧

優先順位	演算子	結合規則	名称
3	(型)	左←右	キャスト演算子
4	* / %	左→右	乗算演算子･除算演算子･剰余演算子
5	+ -	左→右	加算演算子･減算演算子
6	<< >>	左→右	左シフト演算子･右シフト演算子
7	< > <= >=	左→右	左不等演算子　右不等演算子　等価左不等演算子　等価右不等演算子

演算子の一覧

優先順位	演算子	結合規則	名称
8	== !=	左→右	等価演算子 非等価演算子
9	&	左→右	ビット積演算子
10	^	左→右	ビット差演算子
11	|	左→右	ビット和演算子
12	&&	左→右	論理積演算子

演算子の一覧

優先順位	演算子	結合規則	名称
13	||	左→右	論理和演算子
14	?:	左←右	条件演算子
15	= +=	左←右	単純代入演算子 複合代入演算子
16	,	左→右	順次演算子

各種演算子

- 増分・減分演算子
  - オペランド(演算対象)の値を1だけ増・減
  - ++ --
  - c++, ++c は意味が違う!!
- 記憶量演算子
  - 型・定数・関数引用などに対して，記憶域に確保される領域の大きさをバイト単位で返す
  - sizeof(int), sizeof(float), sizeof(double), sizeof(100), sizeof(sin)

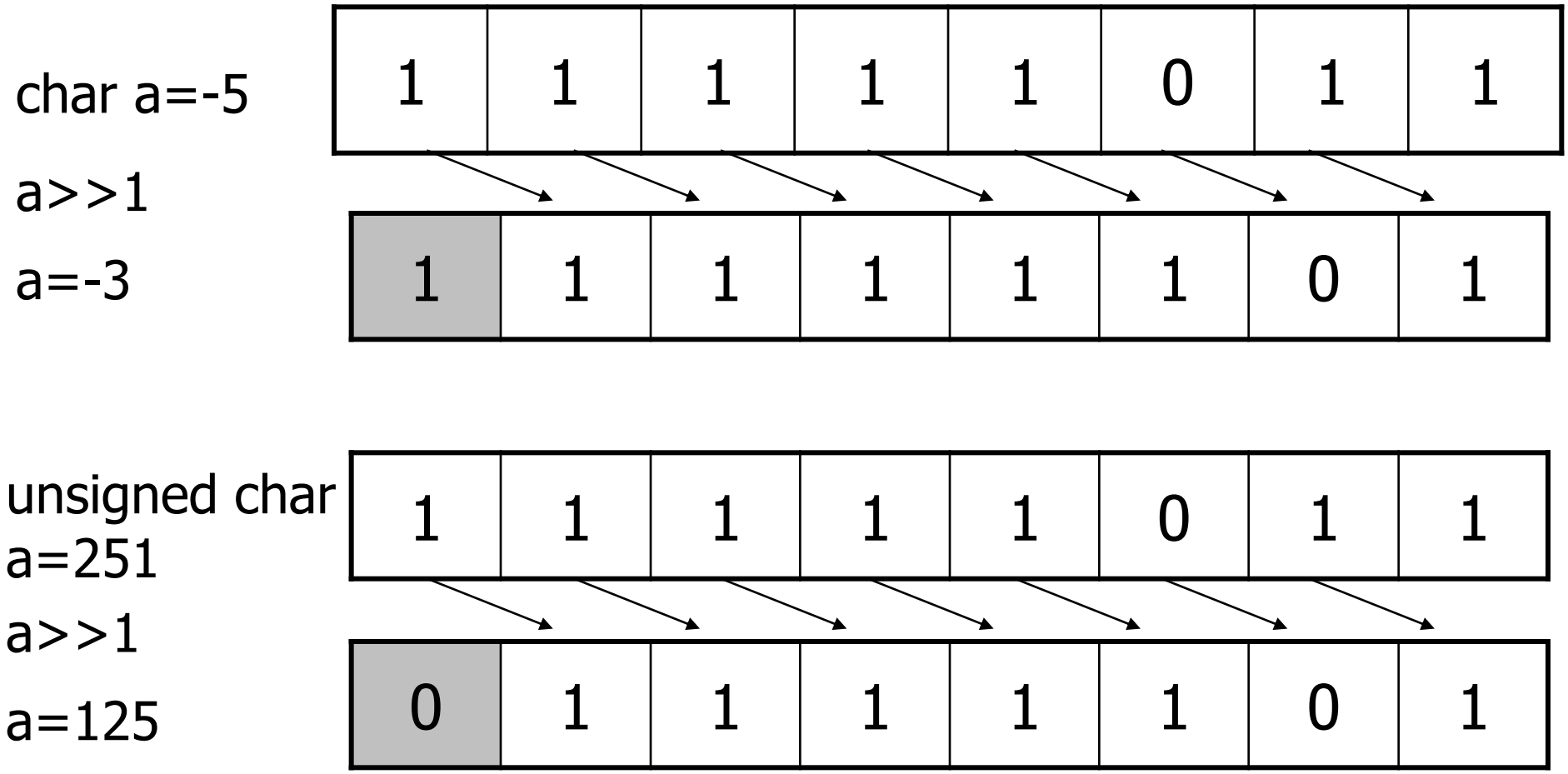
各種演算子

- 算術演算子
  - +, *, -, /, %
  - %は剰余 → x % y：yをxで割った時の余り
- ビット論理演算子
  - 整数値をオペランドとし各ビットに対して論理演算を行う
  - ~ (NOT), | (OR) , & (AND), ^ (XOR)
  - unsigned char a=0x8f, b=0xaa, c;
    c = a^b;

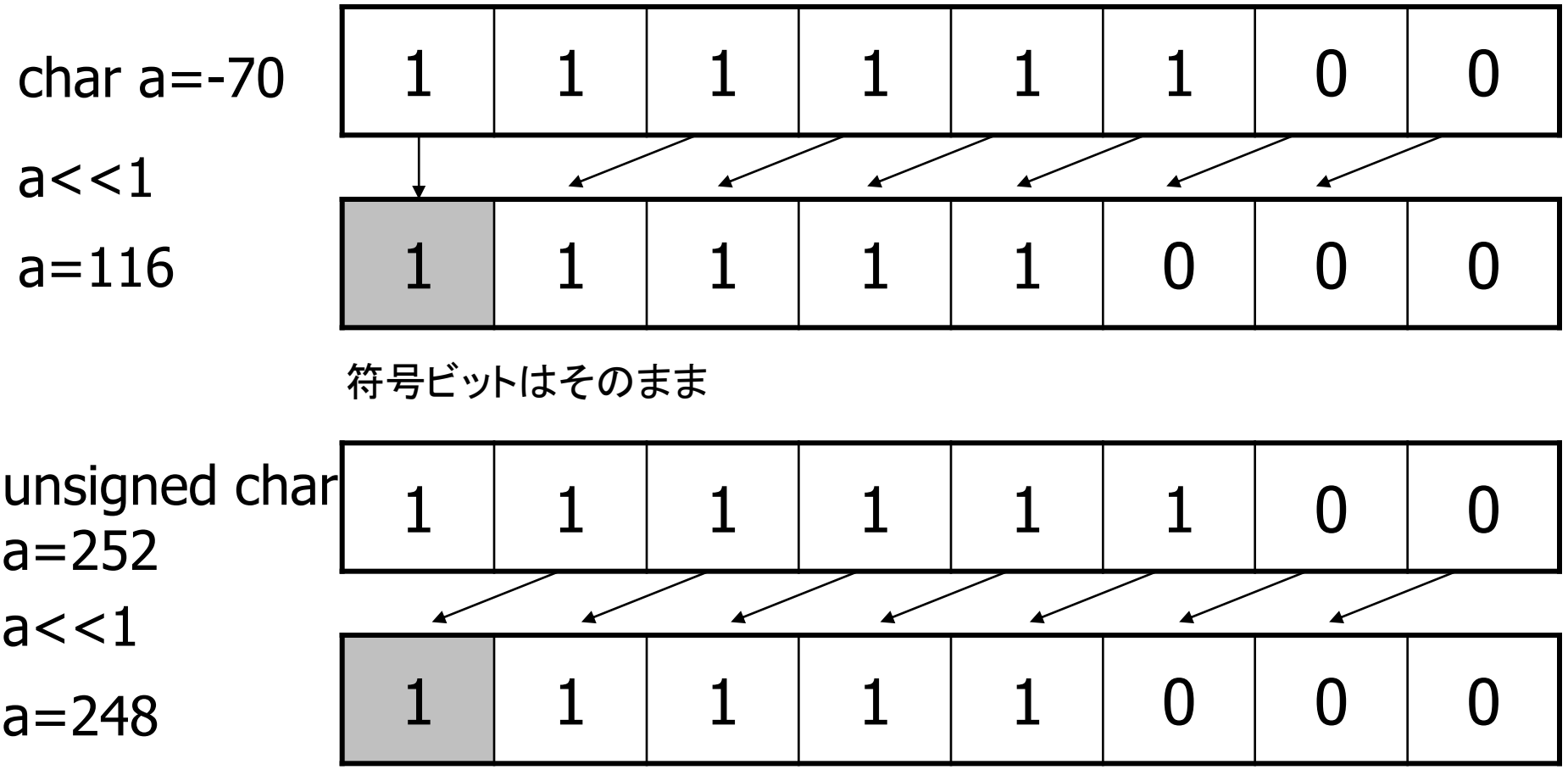
各種演算子

- ビットシフト演算子
  - 第1オペランドの整数値2進数表現を第2オペランドで指示されたビット数だけ左または右にシフト
  - >> 右シフト演算子　<<　左シフト演算子
  - unsinged {char, short, int} は論理シフト
  - signed {char, short, int} は算術シフト
    - 通常singedは記述されない
  - 1ビット右にシフトすると→2で割る
  - 1ビット左にシフトすると→2をかける

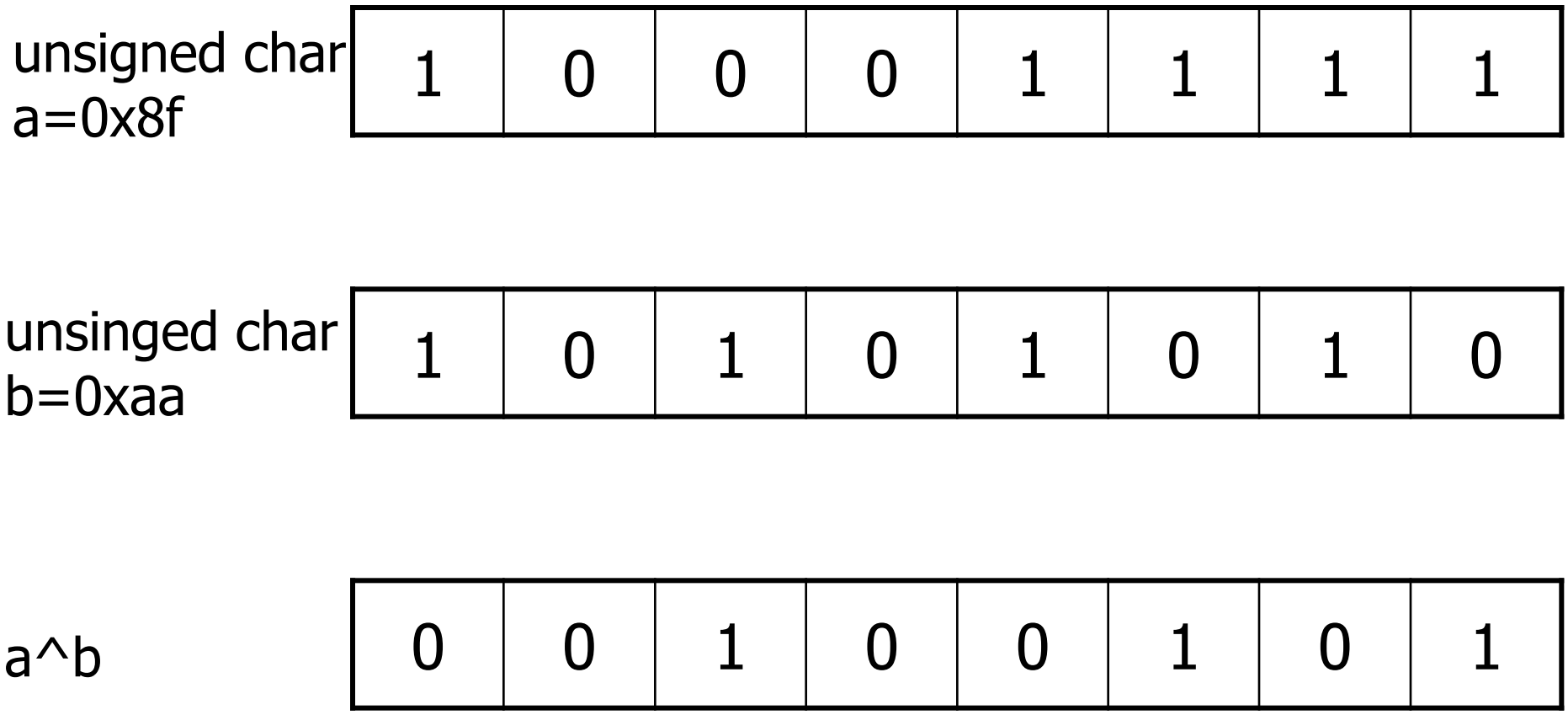
算術シフトと論理シフト



算術シフトと論理シフト



ビット演算子


unsigned char
a=0x8f
1 0 0 0 1 1 1 1
unsinged char
b=0xaa
1 0 1 0 1 0 1 0
a^b
0 0 1 0 0 1 0 1

論理演算

A	B	A&B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A|B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A^B
0	0	0
0	1	1
1	0	1
1	1	0

各種演算子

- 不等演算子
  - 2つのオペランドの大小比較
  - >, >=, <, <=
- 論理演算子
  - 式を論理結合
  - && 結合されたすべての式が非0の時1
  - || 結合されたすべての式が一つでも非0の時1
  - ! オペランドの値が0であるとき評価値1

各種演算子

- 条件演算子 (3項演算子)
  - <条件> ? <式1> : <式2>
  - <条件>が成立したときに<式1>の評価結果を，成立しないときは<式2>の評価結果を値とする
  - int x=1, y=2, z;
    x = (x>y) ? x : y;
- 複合代入演算子
  - i = i +2 → i+=2 と記述可能
  - +, -, *, /, %, <<, >>, &, ^, |

配列とは？ｗ

- 「同じ」データ型の要素を記憶域上に連続して並べたもの
- 配列の要素は添字をつけてアクセス

int data[5];	
data[0]	1
data[1]	2
data[2]	4
data[3]	5
data[4]	20

float array[4];	
array[0]	0.1
array[1]	2.4
array[2]	4.2
array[3]	5.6

なぜ配列が必要か?

- 多量のデータを一括して管理・処理に不可欠

```
int v0, v1, v2, v3, v4;
int sum=0;
scanf("%d", &v0);
scanf("%d", &v1);
scanf("%d", &v2);
scanf("%d", &v3);
scanf("%d", &v4);
sum=v0+v1+v2+v3+v4;
```

```
int v[5], i
int sum=0;
for(i=0;i<5;i++){
        scanf("%d", &v[i]);
}
for(i=0;i<5;i++){
    sum+=v[i];
}
```

要素数が100,200,1000個となった場合にはどちらが楽か?

配列の宣言法 (1次元配列)

- <要素の型> <配列名>[<要素数>];
  - int array[5];
  - float data[10];
  - int array[4]={5,3,2,1};
  - int array[]={5,3,2,1};
  - unsigned char strings[200];
- 注意事項
  - 添字は0から始まる (1からではない!!)
  - 要素数を省略して初期化した場合には初期値の個数が要素数となる

配列要素の取り扱い方

- 要素番号を[]括弧で囲んで配列名に続ける
- 要素は0から始まるのでdata[2]は3番目の要素
- 例
  - int a[5], x, i=1;
  - a[0]=1;a[1]=2;a[3]=5;a[4]=6;
  - x=a[1];
  - x=a[i+1];

配列要素の和を求める (lec5-sum.c)

```
#include <stdio.h>
int getSum(int a[], int num);
main()
{
        int val[8];
        int num=0;
        while(scanf("%d",&val[num])!=EOF){
                num++;
        }
        printf( "Sum = %d", getSum(val,num));
}
int getSum(int a[], int num)
{
        int i, sum=0;
        for(i=0;i<num;i++){
                sum+=a[i];
        }
        return sum;
}
```

1. 関数の仮引数として配列を利用するときは配列の要素数を省略可
2. 要素数はわからないので別途与える必要がある
3. 用意された配列サイズをオーバーするとSegmentation Faultが発生

成績ヒストグラム印字 (lec4-hist.c)

```
#include <stdio.h>
main()
{
  int hist[10];
  int i,j,s,t;
  for(i=0;i<10;i++){
    hist[i]=0;
  }
  while(scanf("%d",&t)!=EOF){
    s = t/10;
    if(s>9) s=9;
    if(s<0) s=0;
    hist[s]++;
  }
  printf( "Cell       ");
  for(i=0;i<10;i++){
    printf("%d ",i);
  }
  printf("¥n");
  printf( "Histogram ");
  for(i=0;i<10;i++){
    printf("%d ",hist[i]);
  }
  printf("¥n");
  printf( "Histogram¥n");
  for(i=0;i<10;i++){
    printf("%d  ",i);
    for(j=0;j<hist[i];j++){
      printf("*");
    }
    printf("¥n");
  }
}
```

読み込んだ点数から級を求め
各級に対応する配列の要素を
1だけ増加

多次元配列

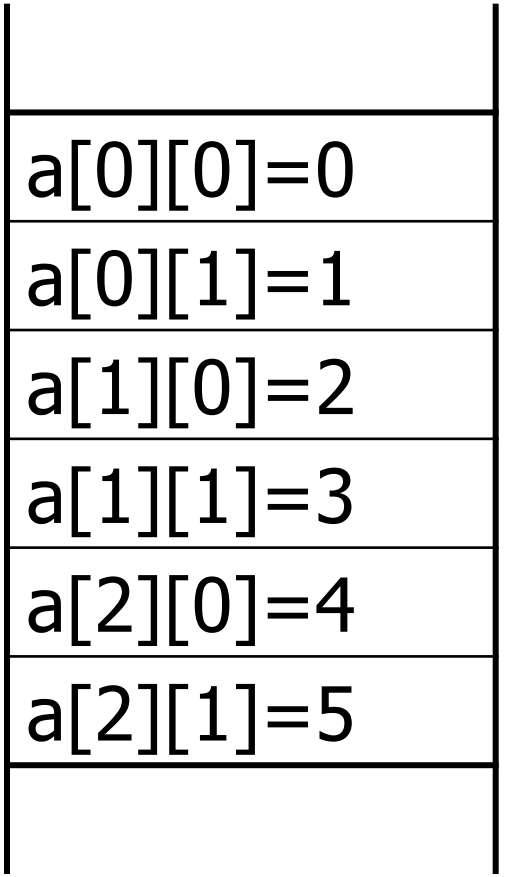
- Cではn次元の配列を取り扱うことができる
- 宣言法
  - <要素の型><配列名>[<第1添字の要素数>][<第2添字の要素数>][<第3添字の要素数>]…[<第n添字の要素数>];
- 例
  - int vals[10][2];
  - int a[3][2]={{0,1},{2,3},{4,5}};
  - int a[][2]={{0,1},{2,3},{4,5}};
  - int a[3][2][2]={{{0,1},{2,3}}, {{0,1},{2,3}}, {{0,1},{2,3}}}; (a[][2][2]は可, a[][][2]は不可)

多次元配列とメモリ

```
int a[3][2]={{0,1},{2,3},{4,5}};
```

a[0][0]=0	a[0][1]=1
a[1][0]=2	a[1][1]=3
a[2][0]=4	a[2][1]=5

構造



メモリにおける配置

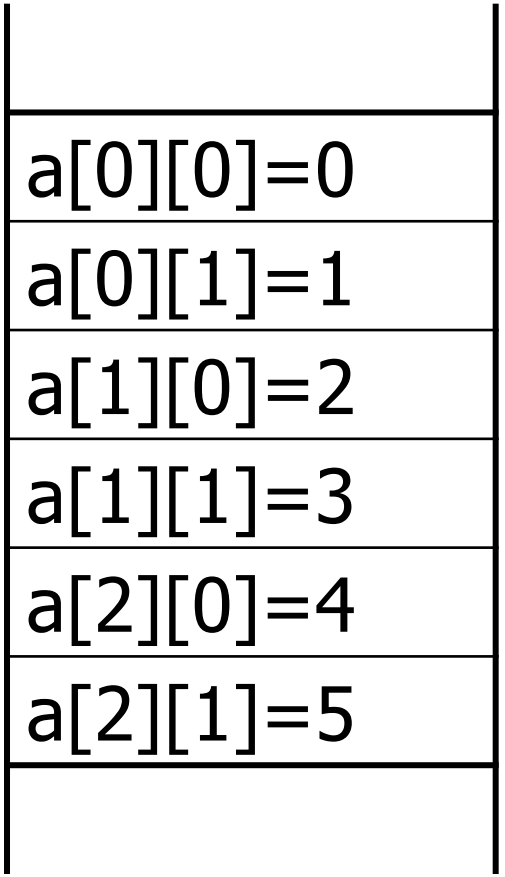
多次元配列とメモリ

- 最後の添字から順にメモリ上に配置される

int a[3][2]={{0,1},{2,3},{4,5}};

a[0][0]=0	a[0][1]=1
a[1][0]=2	a[1][1]=3
a[2][0]=4	a[2][1]=5

構造



メモリにおける配置

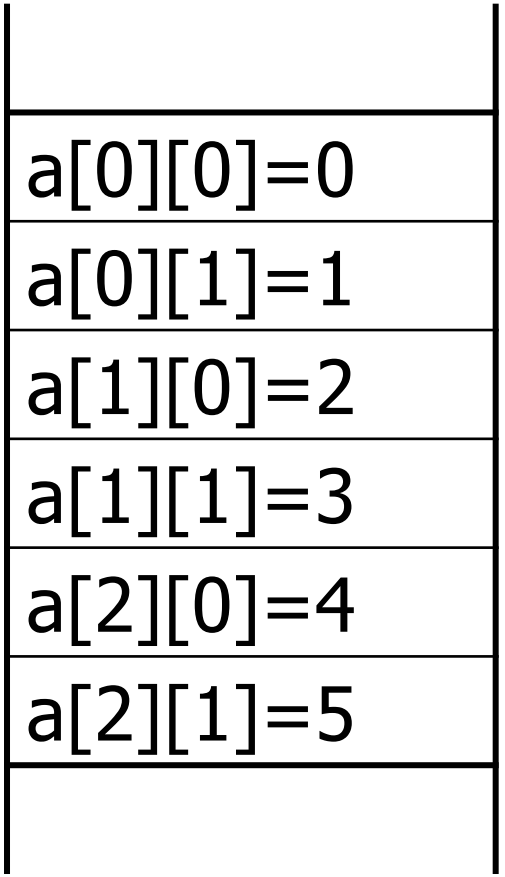
多次元配列とメモリ

```
int i,j,sum=0;
int a[3][2]={{0,1},{2,3},{4,5}};
for(i=0;i<3;i++){
    for(j=0;j<2;j++){
        sum+=a[i][j];
    }
}
```

```
int i,j,sum=0;
int a[3][2]={{0,1},{2,3},{4,5}};
for(j=0;j<2;j++){
    for(i=0;i<3;i++){
        sum+=a[i][j];
    }
}
```

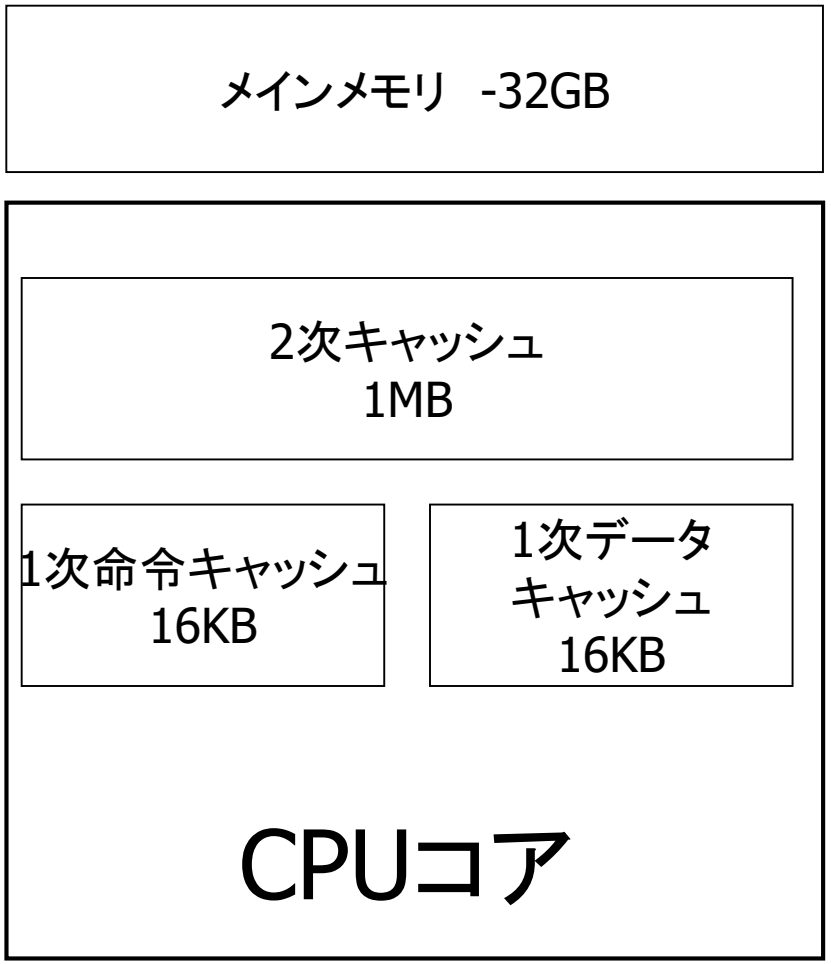
アクセス順序を比較

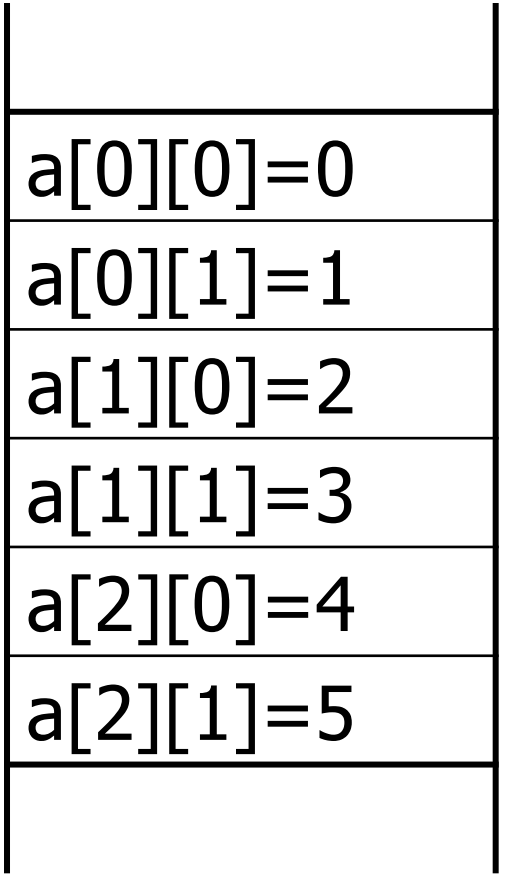
最後の添字から順にメモリ上に配置される



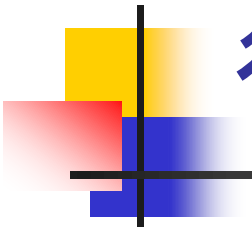
メモリにおける配置

配列アクセスとメモリアクセス





メモリにおける配置



行列の積

行列AとBの積を計算して行列Cに代入

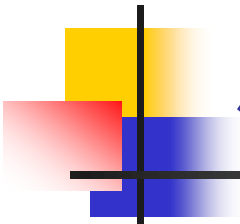
```
for(行列Cの全ての行について繰り返し. 行の番号をi){
    for(行列Cの全ての列について繰り返し. 列の番号をj){
        c_ijを0で初期化
        for(行列Aの全ての列について繰り返し.列の番号をk){
          c_ij += a_ik *b_kj
        }
    }
}
```

行列の積を求めるプログラム（lec5-mul.c)

```
#include <stdio.h>
void subMatMul(int x[][3], int y[][2], int z[][2], int row);
main()
{
    int a[3][3]={{2,1,1},{1,2,1},{1,1,2}};
    int b[3][2]={{2,1},{2,1},{2,3}};
    int c[3][2];
    subMatMul(a,b,c,3);
    …
}
void subMatMul(int x[][3], int y[][2], int z[][2], int row)
{
    int i,j,k;
    for(i=0;i<row;i++){
        for(j=0;j<2;j++){
            z[i][j] =0;
            for(k=0;k<3;k++){
                z[i][j]+=x[i][k]*y[k][j];
            }
        }
    }
}
```

このプログラムではn行3列x3行2列の行列の積しか計算できず

p行q列xq行s列の行列を取り扱うことができるようにするには更なる学習(ポインタ)が必要



注意事項

- Cでは配列の添え字の範囲がチェックされない
  - int array[20];
    array[20]=5; これはエラー．よくある間違い
- Cでは配列の丸ごと代入はない
  - char a1[10], a2[10];
    a2 = a1; これもエラー．これができると便利だけど．
- 多次元配列を作成するときはメモリオーバーに注意
  - int a[1000][1000][1000]; 何バイト必要?