

夏期 C 言語講習

情報研究会 C A C T A S

動的確保

同じ型のデータをまとめて格納するとき配列を利用してきた

配列ではプログラムを書く時点でデータの個数が分かっているといないとダメであった

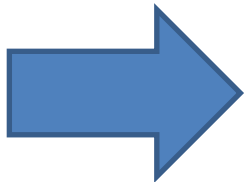
では、扱うデータの大きさがプログラムを書く時点では定まらないときどうすればいいだろう

サンプルプログラム 1

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4
5  int main(void) {
6      int n, i;
7      scanf("%d", &n);
8      double data[n];
9
10     for (i = 0; i < n; i++) {
11         data[i] = 1.0 / (i + 1);
12         printf("%2.5f %n", data[i]);
13     }
14     return 0;
15 }
```

配列の欠点

- 長さは固定である (array[定数])
- 入力はいくらでも大きな値を設定できる
- 十分に大きな領域をとっても入力が小さな値だと無駄な領域をとってしまう.



動的確保

サンプルプログラム 1

出力結果

整数の入力:12

1.00000
0.50000
0.33333
0.25000
0.20000
0.16667
0.14286
0.12500
0.11111
0.10000
0.09091
0.08333

```
1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main(void) {
5     int n, i;
6     printf("整数の入力:"); scanf("%d", &n);
7
8     //メモリの動的確保
9     double *data = (double *)malloc(sizeof(double)*n);
10    if (data == NULL) exit(0);
11
12    //実数列 a_n=1/n (n>0);
13    for (i = 0; i < n; i++) {
14        data[i] = 1.0 / (i + 1);
15        printf("%2.5f %n", *(data + i));
16    }
17
18    //メモリの解放
19    free(data);
20    return 0;
21 }
22
```

sizeof演算子

- sizeofに渡された型や変数のメモリサイズを調べるもの

例

sizeof(int)	=>	4バイト
sizeof(double)	=>	8バイト
sizeof(char)	=>	1バイト
sizeof(char *)	=>	4バイト

malloc()

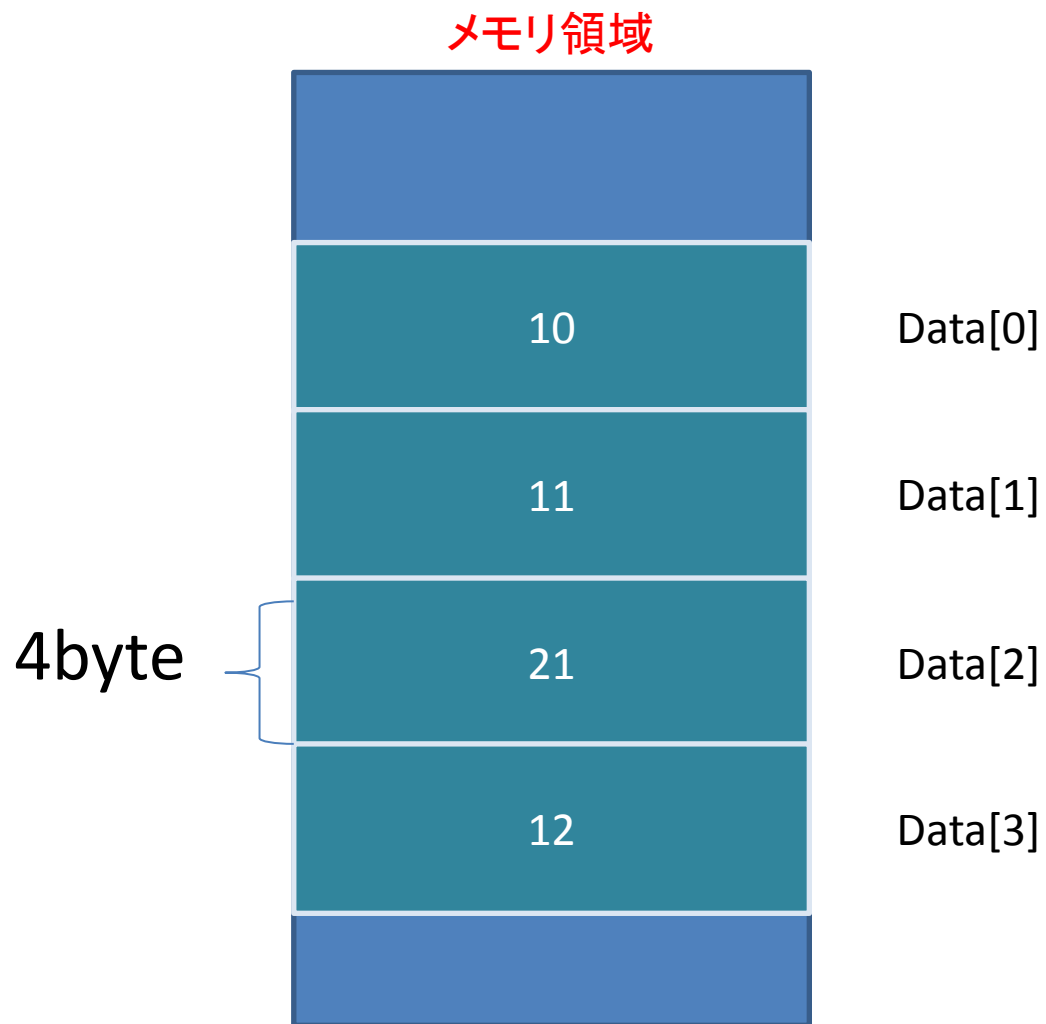
書式	void* malloc(size_t size)
機能	動的メモリ領域の確保をする
引数	size_t size : 領域を確保する為のバイト数を指定します
戻り値	成功すると、確保した領域の先頭ポインタを返し 失敗すると、NULLを返します

メモリとは

- どんなPCにでもついている必須部品です
- 有限な**メモ**リなので節約して使う必要がある
- プログラム, 変数, 定数が実際に保存されている場所

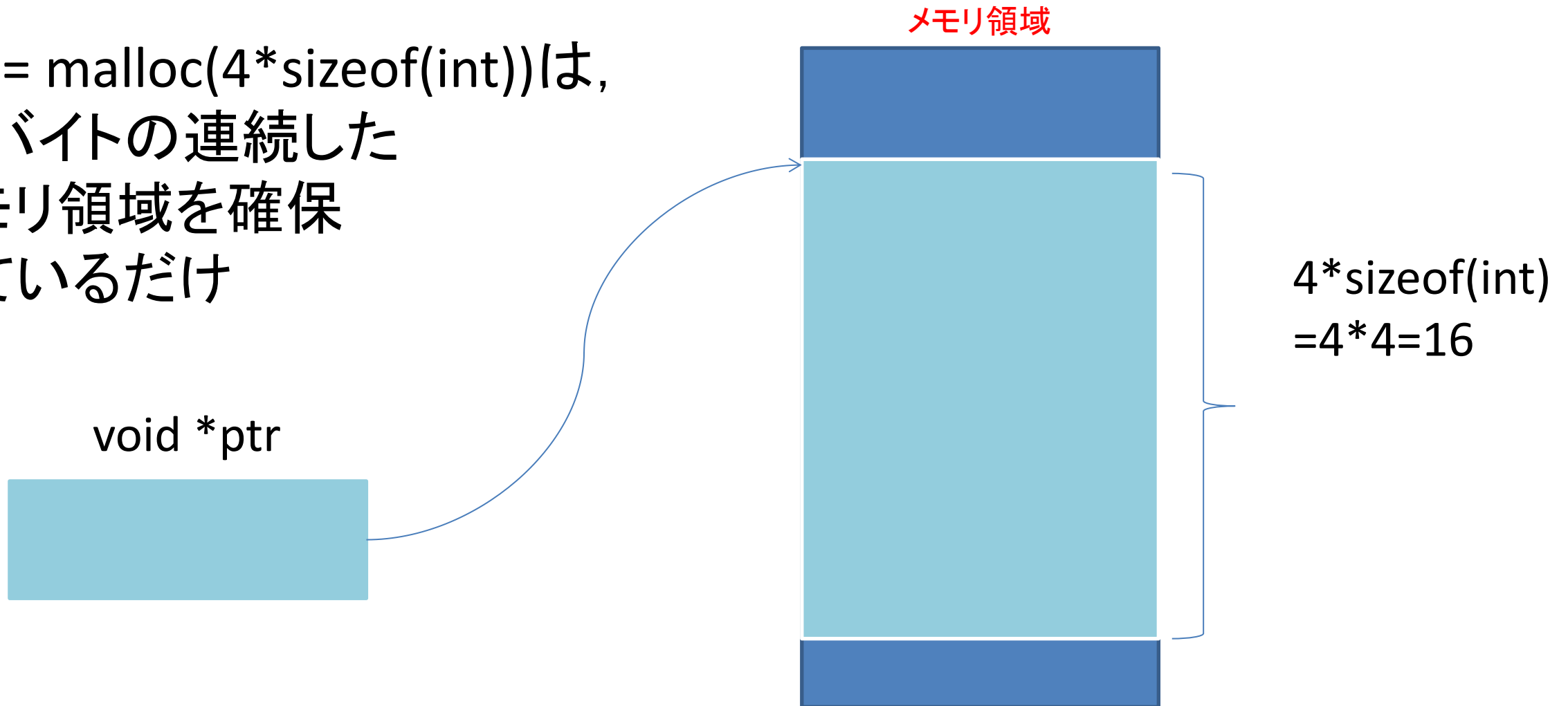
配列のイメージ

int Data[4]は, int型の
4つの入れ物を確保する



mallocのイメージ

`ptr = malloc(4*sizeof(int))`は、
16バイトの連続した
メモリ領域を確保
しているだけ

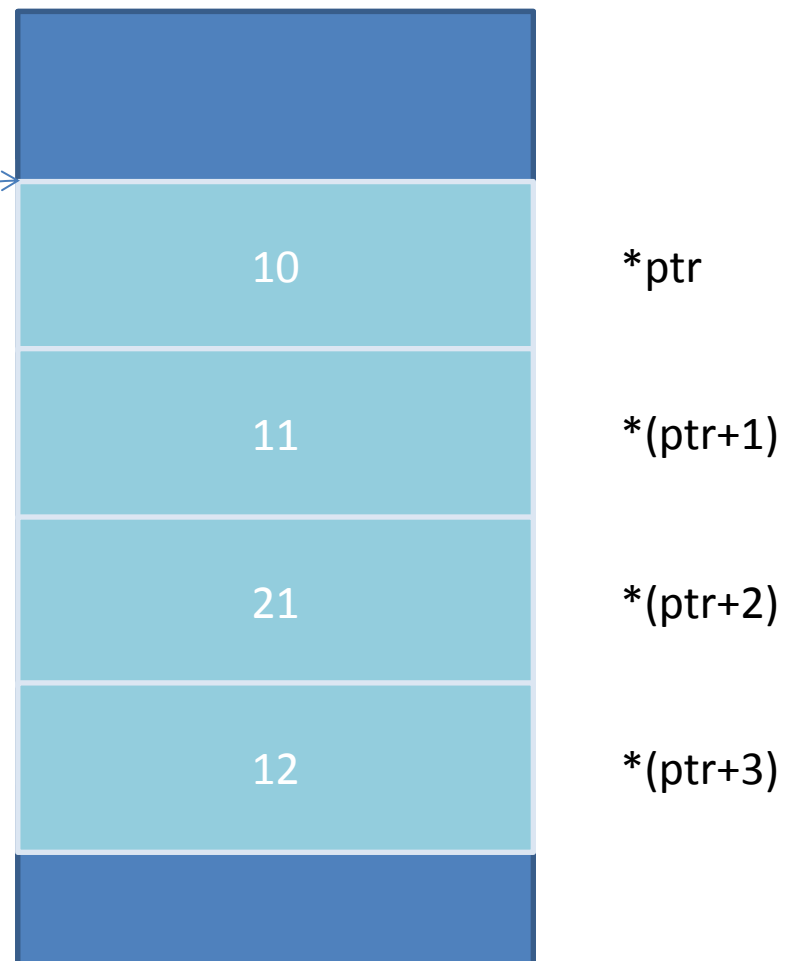


mallocのイメージ

`ptr = (int *)malloc(4*sizeof(int))`
キャスト変換でまるで
配列のように
扱うことができる



メモリ領域



free()

書式	void malloc(void *ptr)
機能	動的に割り当てたメモリ領域の解放をする
引数	void *ptr : 解放するメモリブロックのポインタ
戻り値	なし

free()

- malloc等で動的確保した領域を解放するための関数
- “**使い終わったおもちゃはおもちゃ箱に戻す**”の精神
- 解放された領域は再び動的確保されるために使うことができる

練習問題1

- データ数を入力した後，整数値をその数の分だけ入力し逆順に出力するプログラムを作成しなさい
ただし，プログラム中にmallocを用いること

```
データ数の入力:6  
あと6個入力してください 10  
あと5個入力してください 6  
あと4個入力してください 4  
あと3個入力してください 12  
あと2個入力してください 4  
あと1個入力してください 90  
90 4 12 4 6 10
```

サンプルプログラム 2

出力結果

二つの整数の入力: 8 4

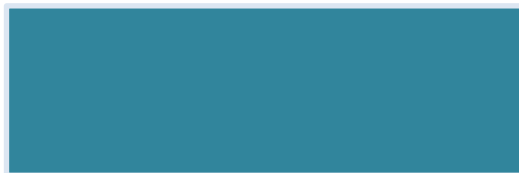
1	2	3	4
2	4	6	8
3	6	9	12
4	8	12	16
5	10	15	20
6	12	18	24
7	14	21	28
8	16	24	32

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  int main(void) {
5      int n,m, i,j;
6      printf("二つの整数の入力:"); scanf("%d%d", &n,&m);
7
8      //メモリの動的確保
9      int **data;
10     data = (int **)malloc(sizeof(int *)*n);
11     if (data == NULL) exit(0);
12
13     for (int i = 0; i < n; i++) {
14         data[i] = (int *)malloc(sizeof(int)*m);
15         if (data[i] == NULL) exit(0);
16     }
17
18     for (i = 0; i < n; i++) {
19         for (j = 0; j < m; j++) {
20             data[i][j] = (i+1)*(j+1);
21             printf("%3d", data[i][j]);
22         }
23         printf("\n");
24     }
25
26     //メモリの解放
27     for (i = 0; i < n; i++) free(data[i]);
28     free(data);
29     return 0;
30 }
```

mallocのイメージ 2

Int **dataは, int型のポインタを
指すポインタである

Int **data




メモリ領域



mallocのイメージ 2

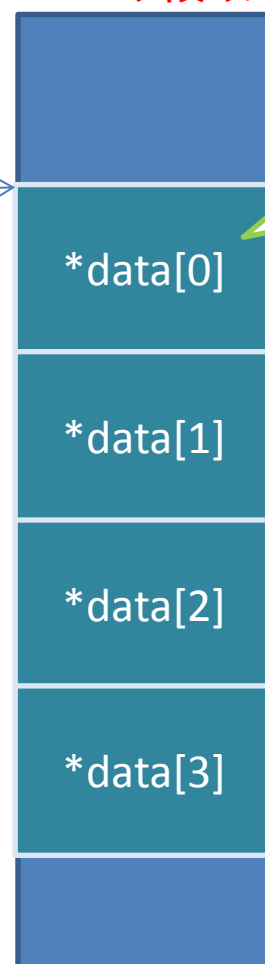
`data=(int **)malloc(sizeof(int *) *n)`
はint型のポインタをn個分
動的確保している

Int **data



A teal rectangular box representing a memory block. A blue curved arrow originates from its right side and points to the first element of a vertical stack of memory blocks.

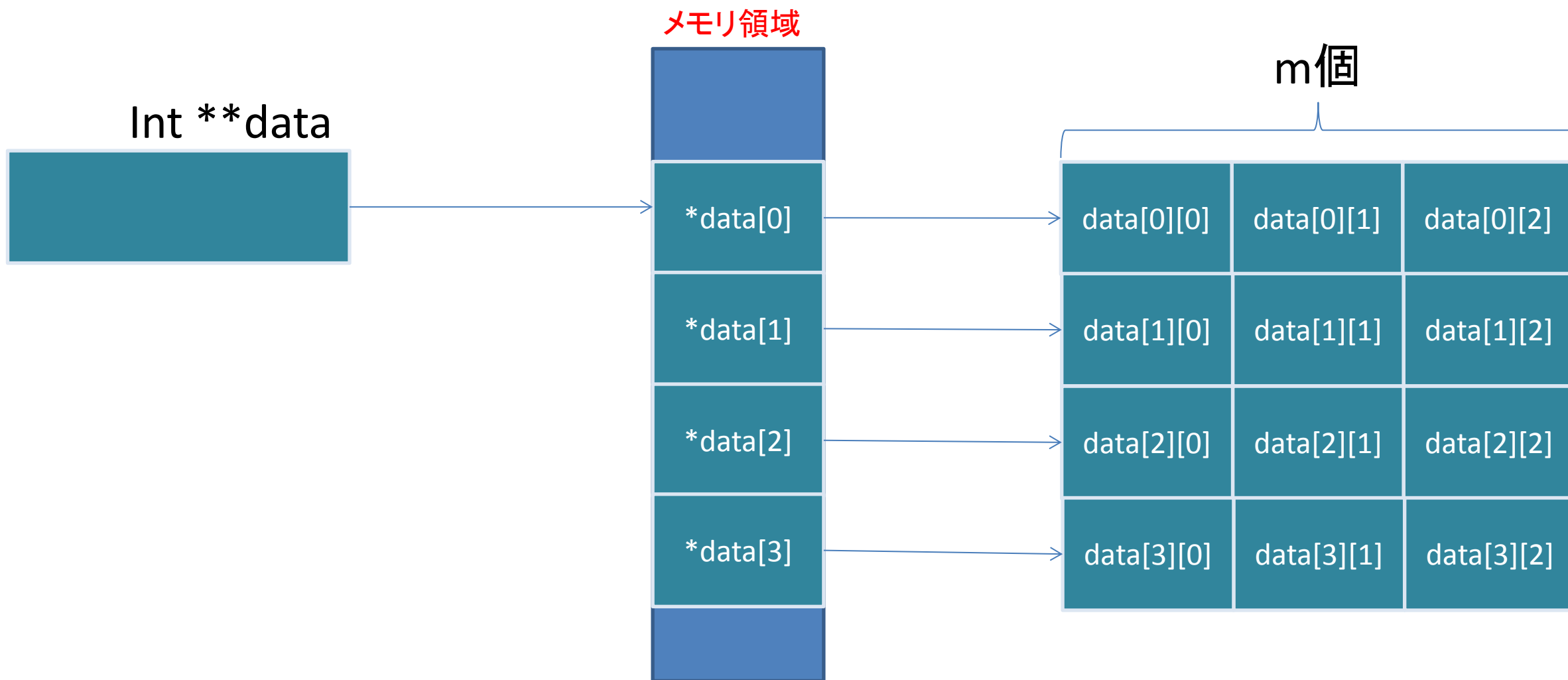
メモリ領域



1つ1つのデータに
はアドレスが格納
されているよ

n個

mallocのイメージ 2



freeのイメージ 2

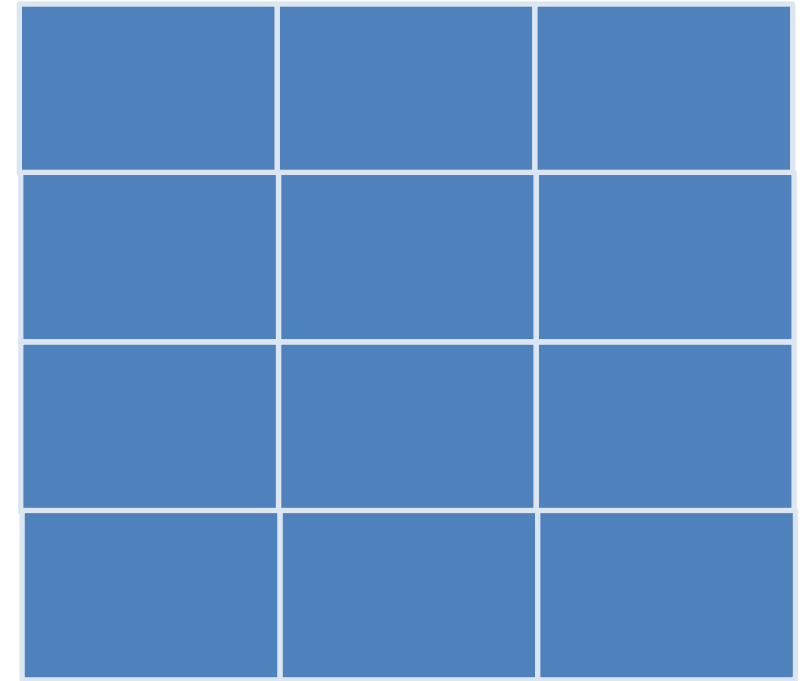
Int **data



メモリ領域



```
//メモリの解放  
for (i = 0; i < n; i++) free(data[i]);
```



freeのイメージ 2

free(data);
で最初に確保したint
型のポインタの配列を
解放する

Int **data



メモリ領域



練習問題2

- 文字列定数buffからn文字分だけコピーする関数MyStrncpyを動的確保を用いて自作しなさい

Char* MyStrncpy(const char *buff,int n)

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  char *MyStrncpy(const char *,int);
5
6  int main(void) {
7      char *ptr = MyStrncpy("LoveGorilla!", 8);
8      printf("%s\n", ptr);
9
10     free(ptr);
11     return 0;
12 }
```

出力結果

LoveGori

realloc()

書式	<code>void realloc(void *ptr, size_t size)</code>
機能	動的に割り当てたメモリ領域を指定したサイズに変更して再度割り当てる.
引数	<code>void *ptr</code> : 現在確保しているメモリブロックのポインタ <code>size_t size</code> : 変更したいメモリのバイトサイズ
戻り値	なし

列挙体の紹介

```
enum test{  
    zero,one,two,three};
```



0

1

2

3

```
#define zero 0
```

のようにプログラム中のどこでもzeroを使うことができる

test x; のように宣言できる. zero,one,two,threeのいずれかの値をとる

列挙体の紹介

```
今日は月曜日です
Sun = 0
Mon = 1
Tues = 2
today = 1
tomorrow = 5
today+1 = 2
nine = 9
ten = 10
eleven = 11
```

```
1  #include<stdio.h>
2
3  enum date{
4      Sun, Mon, Tues, Wednes, Thurs, Fri, Satur};
5  enum num{
6      nine = 9, ten, eleven
7  };
8
9  int main(void) {
10     date today, tomorrow;
11     today = Mon;
12     tomorrow = Fri;
13
14     if (today == Mon) printf("今日は月曜日です\n");
15     if (today + 1 == tomorrow) printf("これは表示されない\n");
16
17     printf("Sun = %d\n", Sun);
18     printf("Mon = %d\n", Mon);
19     printf("Tues = %d\n", Tues);
20     printf("today = %d\n", today);
21     printf("tomorrow = %d\n", tomorrow);
22     printf("today+1 = %d\n", today + 1);
23
24     printf("nine = %d\n", nine);
25     printf("ten = %d\n", ten);
26     printf("eleven = %d\n", eleven);
27
28     return 0;
29 }
```