プログラミング発展

2023年度2Q　火曜日5~7時限(13:45~16:30)
金曜日5~7時限(13:45~16:30)

工学院　情報通信系

尾形わかは，松本隆太郎，
Chu Van Thiem, Saetia Supat
TA:東海林郷志，千脇彰悟

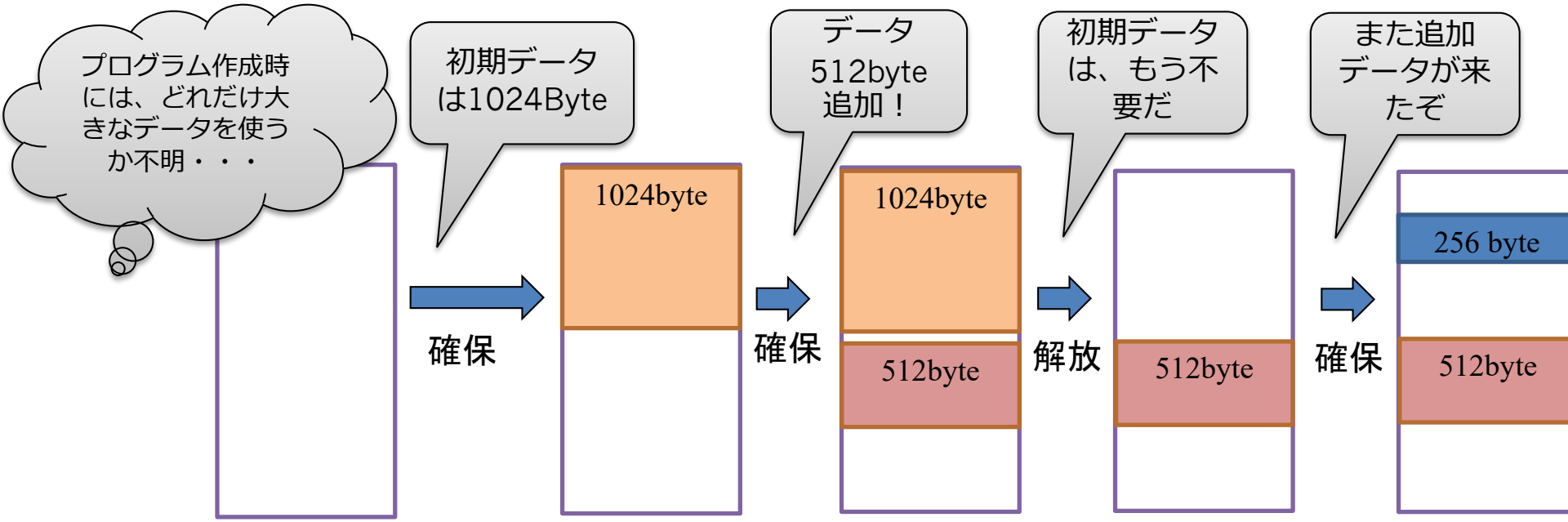
第6回「メモリの動的確保」

1. 前回の復習（課題の解説）
2. 手動でのメモリ確保・解放

手動でのメモリの確保と解放

手動でのメモリの確保

- プログラムの実行中の任意のタイミングでメモリを確保したり開放することが可能
- 一般にこれを「メモリの動的確保」と呼ぶので、本講義でも以降で「動的確保」と言った場合はこれを指す
  - 実際には、自動変数のメモリも動的に確保されているのだが。。。



手動でのメモリ確保（一般に動的確保とよばれるもの）

メモリの動的確保

手動によるメモリの動的確保の利点：大きなメモリが使用可

- **スタック領域**：
  自動変数などが確保される領域（可変長配列も同様）．
  最後に確保したメモリが最初に解放される（スタック）．
  関数呼び出し時にメモリ確保 → returnでメモリ解放など．
  一般に，あまり広くない（macやlinuxだと8MB，windowsだと1MBなど）
- **ヒープ領域**：
  手動によるメモリ確保で使われる領域．
  通常，スタック領域に比べて大きい．ただし，管理は面倒．

- 他に，大域変数や静的変数(static)が置かれる領域もあるが割愛

メモリの動的確保・解放

- 手動でメモリを動的に確保するにはmalloc関数を，
  確保したメモリを解放するにはfree関数を使用する
  - `void *malloc(size_t size);`

万能ポインタ型

    - ヒープ領域のメモリから size バイトのブロックを確保し，
      そのメモリブロックの先頭のアドレスを返す
    - 何らかの理由でメモリが確保できなかった場合，
      NULLを返す
    - 必要なsizeは，sizeof( )関数で計算する
      例： `malloc(sizeof(char)*length);`

→ char型配列（長さがlength）用のメモリを確保

任意のポインタ型を引数に与えてよい．自動的に型変換されるので，
free((void *)ptr); のように型キャストする必要はありません．

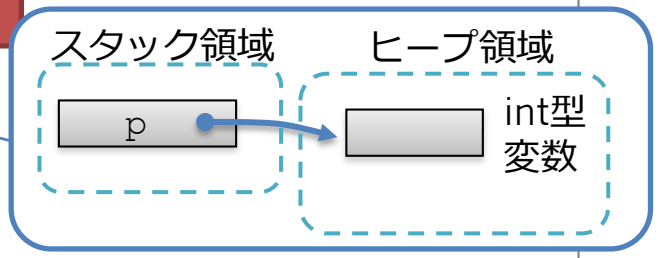
  - `void free(void *ptr);`
    - ptrが指すメモリブロックを解放する

メモリの動的確保の例

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
     int *p, size;

     p = (int *)malloc( sizeof(int) );
     if (p==NULL){
          printf("Cannot allocate memory.¥n");
          return 1;
     }
     *p= . . . . ;
     printf("%d¥n",*p);

     free(p);
     return 0;
}
```

malloc, freeを使用する際には stdlib.h をインクルード

void * から他のポインタ型へは、暗黙のキャストが行われるので、明示的にキャストしなくても良い



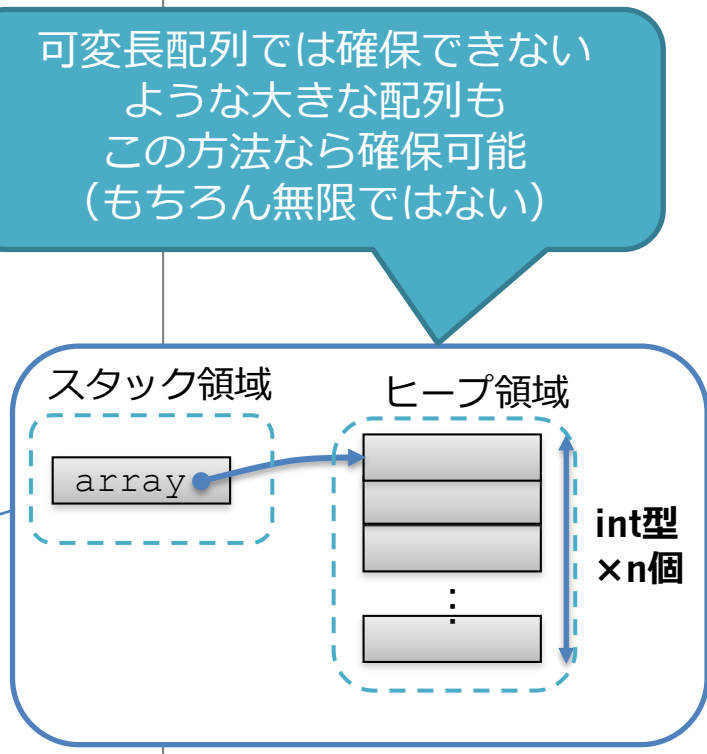
メモリが確保できたかどうかチェック

確保したメモリは使用済みになったら解放する（多くの処理系では、プログラムの終了時に自動的にメモリの解放が行われるが、間違いがないように手動で解放すること）

※説明のためのコード．通常はこのようなことはしないです．

メモリの動的確保の例2

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
     int n,i;
     int *array;

     printf("Input memory size:");
     scanf("%d", &n);

     array = (int *)malloc( sizeof(int)*n );
     if (array==NULL){
          printf("Cannot allocate memory.¥n");
          return 1;
     }
     for (i=0; i<n; i++) {
          array[i] = i*i;      // *(array+i) = i*i; と同じ
          printf("%d¥n",array[i]);
     }
     free(array);
     return 0;
}
```



Ex6-1

- メモリの動的確保の練習.

① 以下のAからGを行うプログラムを作成しなさい.

A) まず,

```
int *a[10];
```

と宣言し, a[0]～a[9]をNULLで**初期化**する.

B) i=0,1,2,3,4に対し, 以下を行う.
int型変数用のメモリを**mallocを使って動的に確保**し,
確保したアドレスをa[i]に入れる.

C) a[0]～a[9]を標準出力に表示する.

D) a[1], a[2], a[4]で示されたメモリ領域を**解放(free)**し,
a[1], a[2], a[4]にNULLを代入する.

E) a[0]～a[9]を標準出力に表示する.

F) i=5,6,7,8,9に対し, 以下を行う.
int型変数用のメモリを**mallocを使って動的に確保**し,
確保したアドレスをa[i]に入れる.

G) a[0]～a[9]を標準出力に表示する.

H) 解放する必要のあるメモリをすべて**解放(free)**する.

（次ページに続く）

Ex6-1 （つづき）

② ①のプログラムを実行し、以下をpdfファイルで報告しなさい。

- 実行結果のスクリーンショット（右の例参照）
- 実行したコンピュータのOSと実行環境（統合環境の名前、コンパイラの名前など、わかる範囲で）
- 確保したメモリは、順番に並んでいるか否か
- freeしたメモリが、その後に再利用されているか否か

※環境等により、確保したメモリは順番に並んだり、ランダムに見える順番だったり、freeしたところが優先的に再利用されたり、そうでなかったりする。自分の実行結果を見て判断すること。

★提出物：ex6_1.pdf

実行例）

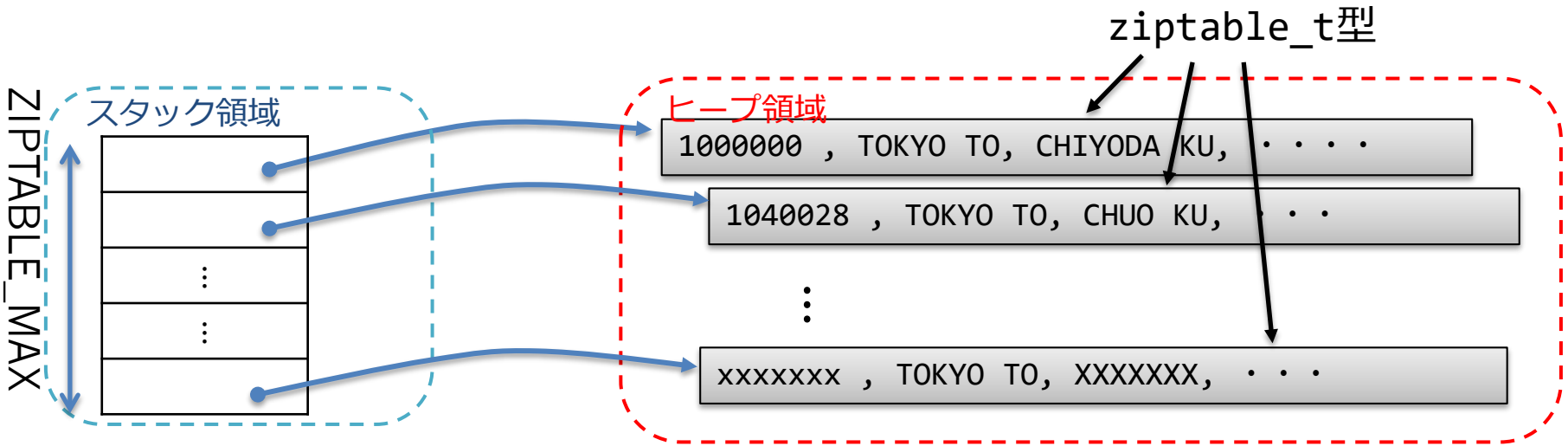
```
C)
a[0] = 0x. . . .
a[1] = 0x . . . .
  :
  :
a[9] = ….

E)
A[0] = 0x. . . .
A[1] = . . . .
  :
  :
a[9] = . . . .

G)
a[0] = 0x. . . .
a[1] = . . . .
  :
  :
a[9] = . . . .
```

Ex6-2

Ex4-2で用いた郵便番号データを考える．Ex4-2では，データを大きな配列 ziptable_t ziptable[ZIPTABLE_MAX] に保存した．

しかし，ZIPTABLE_MAXが大きくなると，スタック領域に入りきらなくなる恐れがある．また，ヒープ領域であっても，巨大な配列（一塊のメモリ領域）を確保するのは難しいかもしれない．

そこで，データ本体は，ヒープ領域に1レコードごとに保存することにし，各レコードを保存した場所（アドレス）を別途，配列に保存することにする．

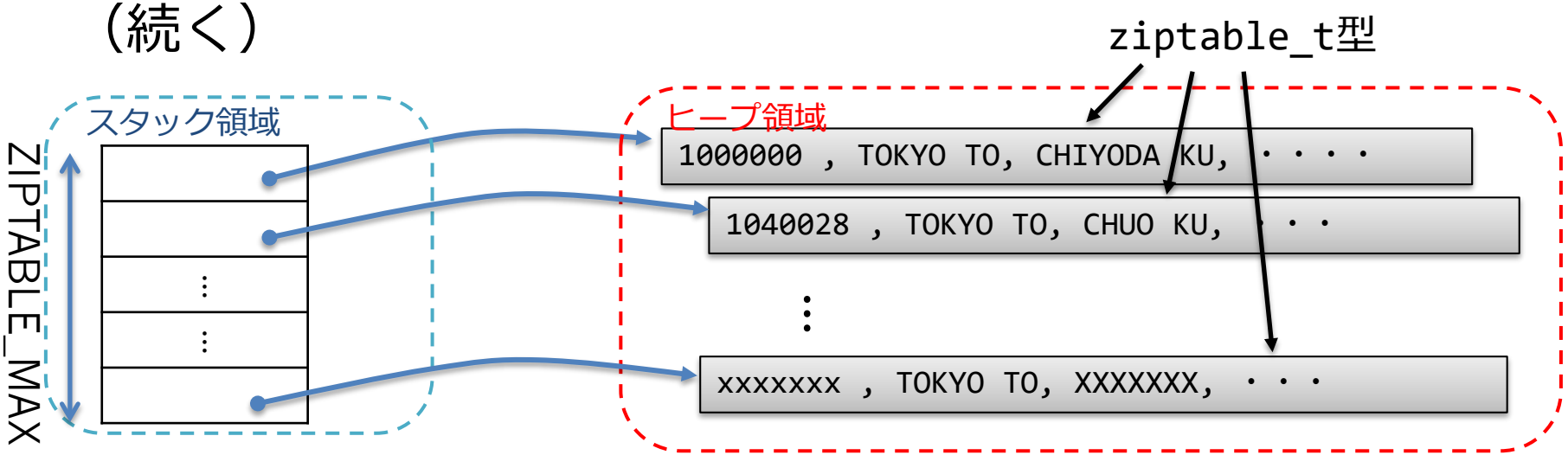


Ex6-2 （つづき）

① 以下の動作をするプログラムを作りなさい.

1. スタック領域に，ziptable_t用のポインタ（ziptable_t *）をZIPTABLE_MAX個格納する配列を用意する.
2. 郵便番号リストのファイルから，1行ずつ読み込み， mallocでziptable_t型変数用のメモリを確保し，確保したアドレスをポインタ格納用の配列に1つずつ格納し，確保したメモリに読み込んだデータを書き込む．ファイルの最後まで読み込んだら，読み込んだデータ数を表示する.
   (Ex4-2で用いた関数　read_from_csv　を変更する)

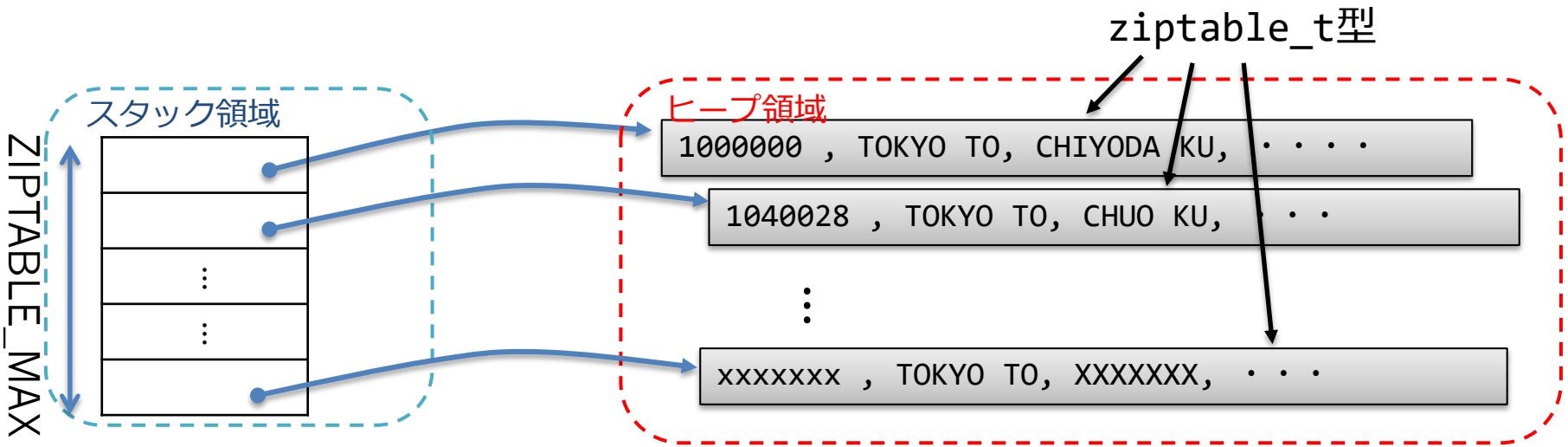
（続く）



Ex6-2 （つづき）

① の続き

3. 標準入力から，郵便番号（十進７桁）を読み込み，線形探索を行うことで，県名，市区名，町名などを出力する.
   探索で該当するデータが見つからない場合は、 no data などと表示する.



Ex6-2 （つづき）

- 構造体ziptable_tは，zip2021.c にある通りとする。

```
#define ZIPCHAR_SIZE 8
#define PREF_SIZE 20
#define CITY_SIZE 50
#define ADDR_SIZE 100

typedef struct _ziptable {
    int zip;
    char pref[PREF_SIZE];
    char city[CITY_SIZE];
    char addr[ADDR_SIZE];
} ziptable_t;
```

実行例：

```
Number of data : xxxxxx
zipcode? : 1000001
address : TOKYO TO CHIYODA KU CHIYODA
```

```
Number of data : xxxxxx
zipcode? : 9876543
no data
```

Ex6-2 （つづき）

- 郵便番号のデータは，tokyo_all_dat.txt （東京都のみ）を使ってください．データ数は 4000件を超えないので，
  ZIPTABLE_MAX=4000
  としてください．

- 読み込むファイル名を chiyoda.txt （千代田区のみ）や KEN_ALL_ROME.txt（全国）に変えて，同様に実行できるか試してみるとよい．その場合，ZIPTABLE_MAXは以下を使ってください．
  - chiyoda.txt の場合，ZIPTABLE_MAX=70
  - KEN_ALL_ROME.txt の場合，ZIPTABLE_MAX=130000

- ソートは不要．その場合，探索に二分探索は使えないので，線形探索してください．ソートして二分探索しても構わないが，ソートや二分探索でミスったら減点されます．

Ex6-2 （つづき）

②

郵便番号リストとして全国のデータが入っている KEN_ALL_ROME.txtを使う場合，データ数は12万件を超えるため，ZIPTABLE_MAX=130000とする.

Ex4-2のように全データを一つの配列に格納する場合，この配列のサイズは何バイトになるか概算せよ．（算出根拠も書くこと）

また，Ex6-2①のプログラムのように，スタック領域にはポインタを格納する配列のみを保存する場合，スタック領域に保存される配列のサイズは何バイトになるか概算せよ．（算出根拠も書くこと）

ヒント：配列の要素数は両方ともZIPTABLE_MAX．配列の各要素の長さが異なる.
sizeofで実際の配列サイズを調べて，概算と整合しているか確認してみるとよい.

注意！

次のようなメモリの不適切な利用は、セキュリティホールになりうるため減点対象です.

（エラーが起きず，結果が正しく出力されたとしても．）

- mallocで確保したメモリ領域をfreeせずにプログラムが終了している．あるいは，一度freeしたメモリを，再度freeする.
- 範囲外アクセス：例えば array[4][4]を宣言して array[4][0]に書き込んだりprintfしたり.
- free(p)した後に *p に書き込んだりprintfしたり.

※実行のたびに挙動が異なる場合，このような間違いがあると思われる.

注意！（つづき）

- free(a);
  は、a で示されたメモリ領域を「もう使いません」と宣言したことになるが、aにはそのメモリのアドレスが記録されたたままであり、freeした後であっても
  *a = xxx;
  によって「もう使わない」はずのメモリに代入すること自体は出来てしまう。そのため、プログラムミスにより想定外の動作が起きるが、ミスに気付きにくい。
- 想定外の操作を防止するには、ex6-1のように、free(a); の後に、
  a=NULL;
  と代入しておくとよい。
- a==NULL であれば、free(a);を複数回実行することによる不都合もおこらない。

バイト数について

メモリのサイズなどを表す場合，

$1\mathrm{kB} = 2^{10}\mathrm{B} = 1024\mathrm{B}$
$1\mathrm{MB} = 2^{10}\mathrm{kB} = 2^{20}\mathrm{B}$
$1\mathrm{GB} = 2^{10}\mathrm{MB} = 2^{30}\mathrm{B}$　（以降，TB, PB, EB, ．．．と続く）

であることが多い．しかし，一般には10進法が用いられ，

$1\mathrm{kB} = 10^{3}\mathrm{B} = 1000\mathrm{B}$
$1\mathrm{MB} = 10^{3}\mathrm{kB} = 10^{6}\mathrm{B}$
$1\mathrm{GB} = 10^{3}\mathrm{MB} = 10^{9}\mathrm{B}$　（以降，TB, PB, EB, ．．．と続く）

である．二進法と10進法との混乱を避けるため，

$1\mathrm{KiB} = 2^{10}\mathrm{B} = 1024\mathrm{B}$
$1\mathrm{MiB} = 2^{10}\mathrm{KiB} = 2^{20}\mathrm{B}$
$1\mathrm{GiB} = 2^{10}\mathrm{MiB} = 2^{30}\mathrm{B}$　（以降，TiB, PiB, EiB, ．．．と続く）

という表記もある．

Ex61-②では，「概算」であるから，$1\mathrm{MB} \fallingdotseq 10^{6}\mathrm{B} \fallingdotseq 2^{20}\mathrm{B}$　で構わない

今日の提出物 まとめ

- Ex5-4: 逆行列を求めるプログラムのソースファイル (ex5_4.c)

- Ex6_1: 以下を含むpdfファイル（ex6_1.pdf）
  - ①で作成したプログラムの実行結果のスクリーンショット
  - 上記を実行したコンピュータのOSと実行環境
  - 確保したメモリは、順番に並んでいるか否か
  - freeしたメモリが、その後に再利用されているか否か
- Ex6_2：
  - ①プログラムのソースファイル （ex6_2_1.c）
  - ②バイト数の概算（ex6_2_2.pdf）

コピペレポートについて

プログラムや考察などが他の提出者と重複している場合、不正とみなして減点および問い合わせをすることがあります