第3週
動的メモリ確保とポインタ配列

動的メモリ確保

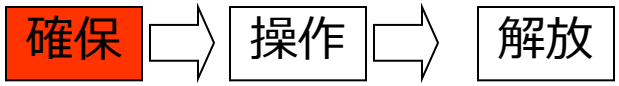
●静的メモリ確保
- 下記のように宣言

```
int x[100];
```

プログラミングする時点で
サイズが分っている場合はこれでよい
各変数が確保すべきメモリ領域の大きさは
既知である必要
配列の場合，配列長も既知である必要
(コンパイル前に決める)

●動的メモリ確保
- プログラムの実行時にメモリを確保すること
- 利点
  - メモリを有効活用できる
    （必要分だけのメモリを確保できる）
- 利用例
  - リスト（７、８週）

動的メモリ確保：malloc



size_t型は符号無し整数型
負の数は受け付けない

```
void *malloc(size_t size);
```

解説：大きさsizeバイトのメモリ領域を確保する

戻り値：確保成功の場合
　　確保したメモリの先頭アドレス
　　確保失敗した場合
　　空ポインタ（NULL）

mallocの使い方

確保 ⇨ 操作 ⇨ 解放

静的メモリ確保

```
A x[B];
```

A：「char」「 int」「 struct 構造体名」などの型名
B：サイズ

動的メモリ確保

代入されるポインタ変数と同じ型にキャストする

```
A *x;
x = (A *) malloc( sizeof(A) *  B  );
```

(型 *) malloc(**sizeof(型)** * **必要な個数**);

sizeof(型)で型のバイトサイズを取得できる
例：sizeof(char)は、１が取得できる

malloc：文字列の例

確保 ⇨ 操作 ⇨ 解放

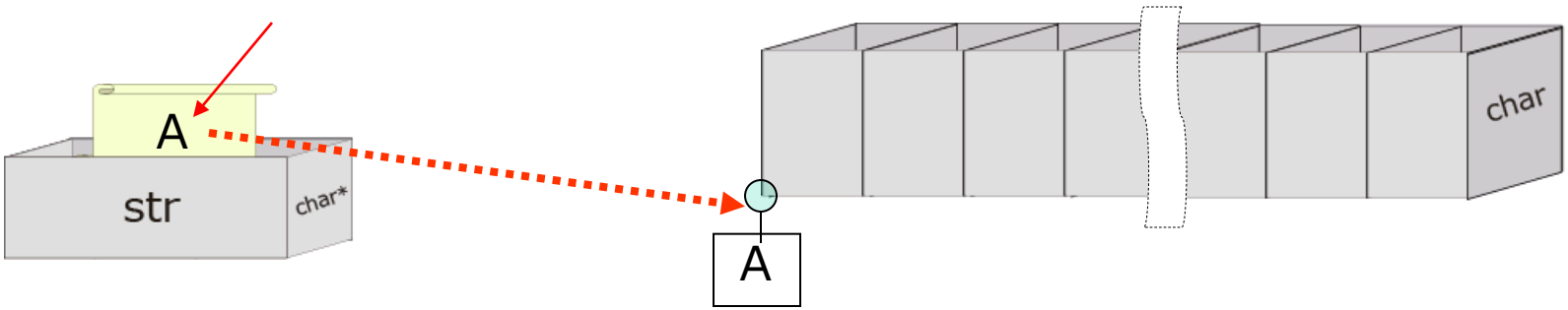
```
char *str;
int size;
...
scanf("%d", &size);
str = (char *) malloc( sizeof(char) * ( size+ 1) );
```

代入されるポインタ変数と同じ型にキャストする

文字列の場合は￥０の分一つ多くメモリを確保する

(型 *) malloc(**sizeof(型)** * **(必要な個数)**);

確保したメモリ領域の先頭アドレス



malloc：int配列の例

確保 ⇨ 操作 ⇨ 解放

```
int *a;
int size;

scanf("%d", &size);
a = (int *) malloc( sizeof(int) *  size );
```

代入されるポインタ変数と同じ
型にキャストする

(型 *) malloc(**sizeof(型)** * **必要な個数**);

```
a[ 0 ] = 0;
```

メモリ確保後は、
配列と同様に利用できる

malloc:エラー処理 (メモリ確保失敗)

確保 ⇨ 操作 ⇨ 解放

A：「char」「 int」「 struct 構造体名」などの型名
B：必要名メモリのサイズ

```
A *x;

x = (A *) malloc(sizeof(A) * B );

if(x == NULL){
  //エラーメッセージ
  //エラー処理
}
```

メモリが確保できなかった場合には
ポインタ変数（x）にNULLが代入されます

free:メモリの解放

確保 ⇨ 操作 ⇨ 解放

A：「char」「 int」「 struct 構造体名」などの型名
B：サイズ

```
A *x;
x = (A *) malloc(sizeof(A) * B );
if(x == NULL){
 //エラーメッセージ
 //エラー処理
}
...
free(x);
```

メモリの利用後は必ずメモリの解放すること
解放したいポインタ変数名を指定する

メモリの解放
free(ポインタ変数名);

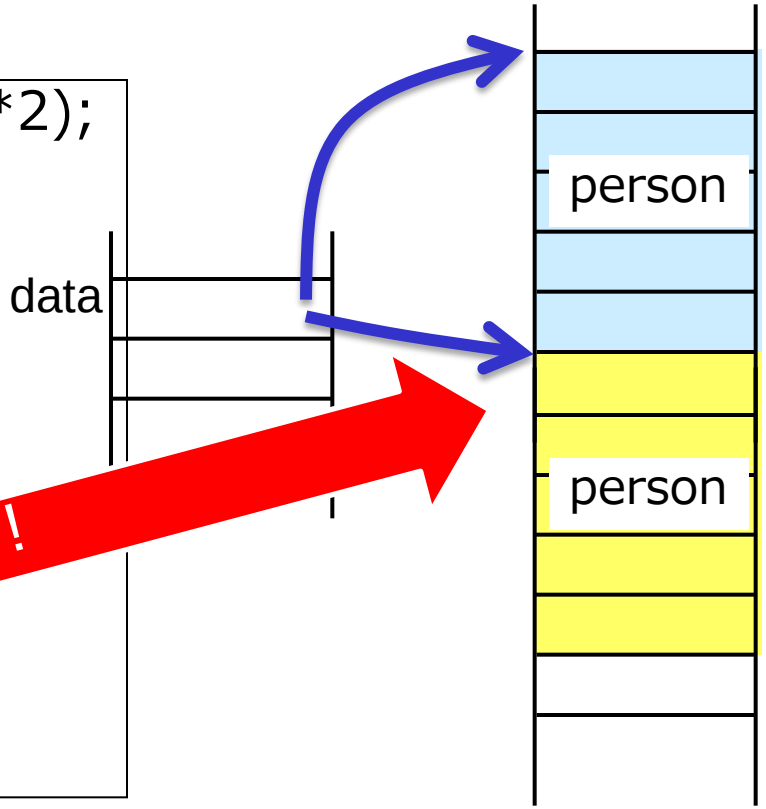
freeでよくする間違い

```
data = (person *)malloc(sizeof(person)*2);

data->age = 30;
strcpy(data->name, "Tomoko Izumi");

data++;
data->age = 40;
strcpy(data->name, "Tar...");

free(data);
```



freeに与える引数は、mallocの戻り値!

必須課題3-1:ヒント（作成手順）

1. コンソールから文字列の最大長（数字）の入力 (scanfなどを利用)

2. 入力された文字列長のメモリを確保する (mallocを利用)

3. 2で確保したメモリ領域に文字列の入力を受け付ける (scanfなどを利用)

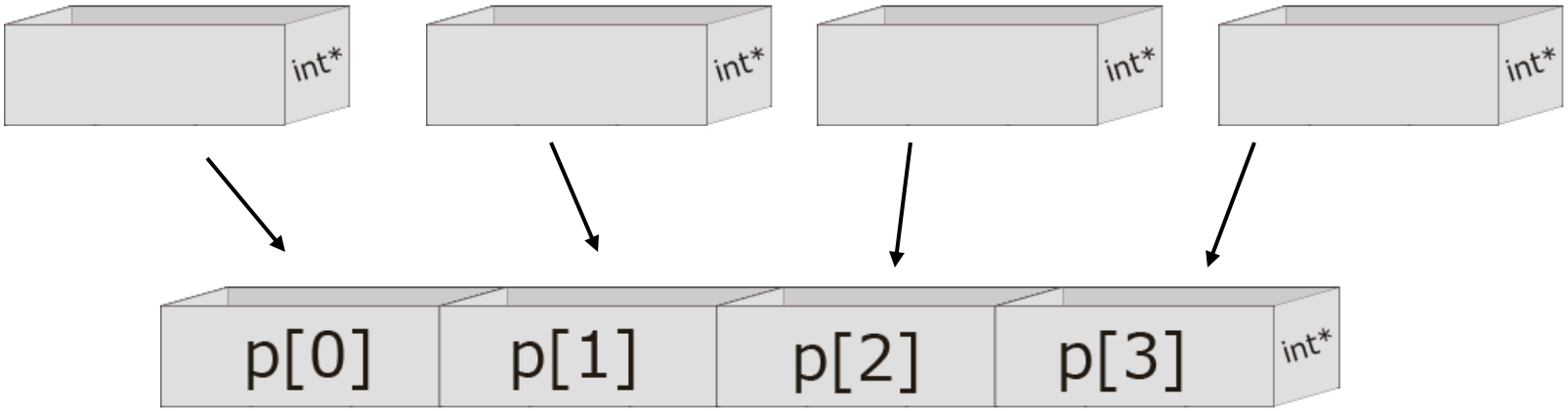
4. 入力された文字列を大文字にして逆順で出力

必須課題3-1:理解補助シート

ポインタ配列

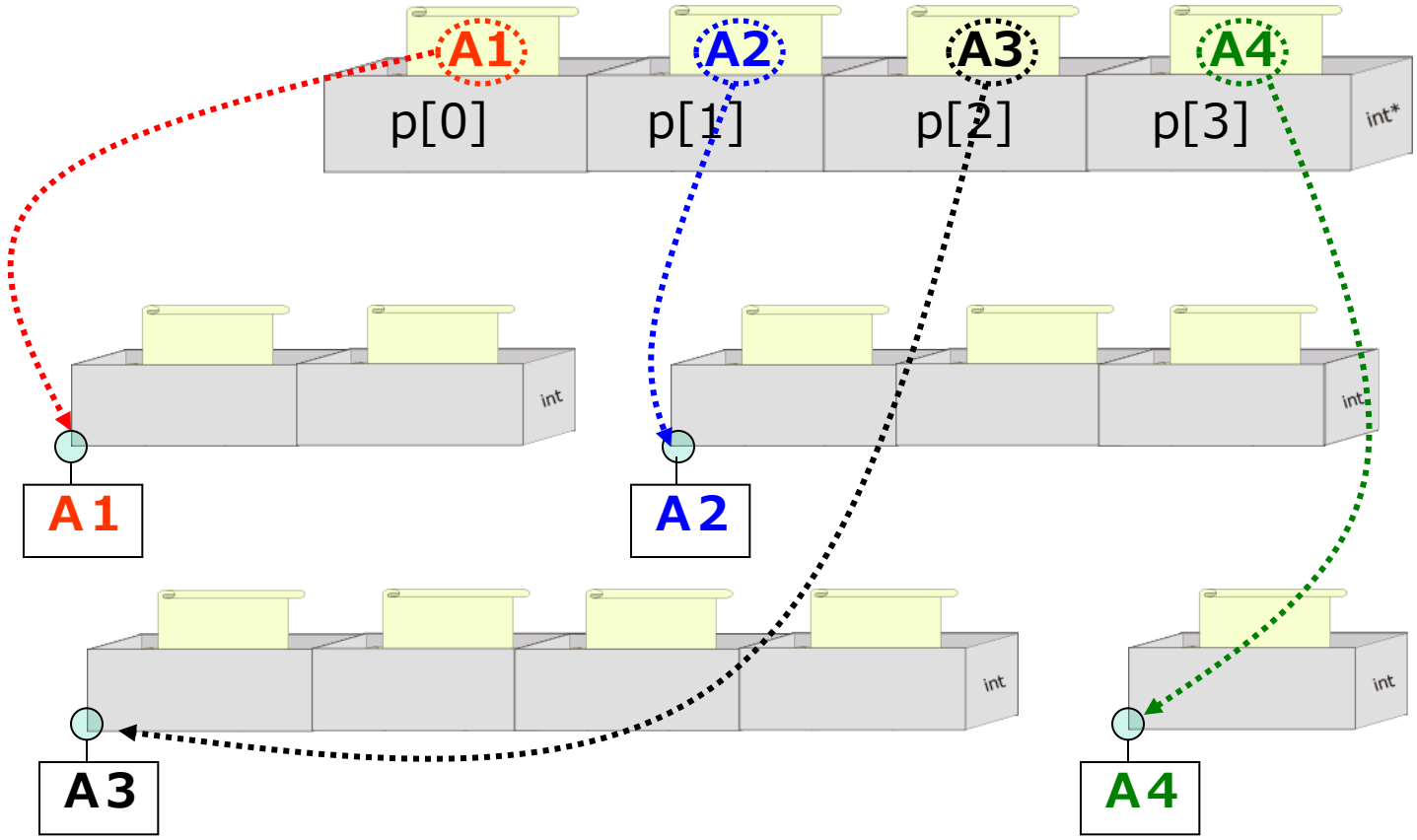
- アドレスを保存している配列
  - 配列
    - 複数の**同じ型**の変数をまとめたもの

```
int     *p[ 4 ];
```



ポインタ配列:intのポインタ配列の例

```
int *p[4];
p[ 0 ] = (int *)malloc( sizeof(int) * 2 );
p[ 1 ] = (int *)malloc( sizeof(int) * 3 );
p[ 2 ] = (int *)malloc( sizeof(int) * 4 );
p[ 3 ] = (int *)malloc( sizeof(int) * 1 );
```



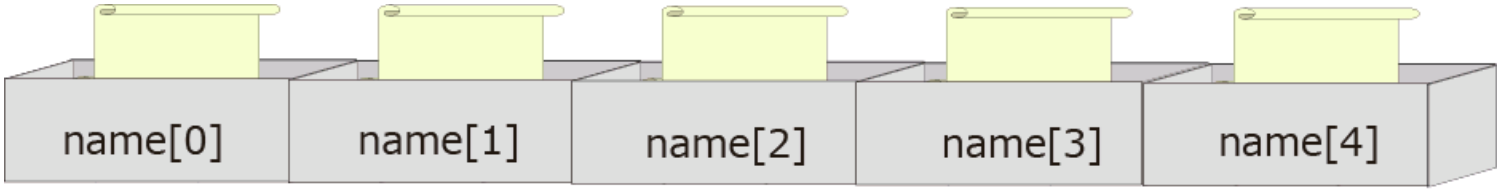
必須課題3-2:ヒント（作成手順）

1. 文字列用のポインタ配列を宣言（char *name[5]）

2. 名前を一人毎入力を受け付ける
   1. 名前を格納できるメモリ領域を確保する
   2. メモリ領域へのポインタを１で宣言したポインタ配列に格納する

3. 入力された人数分下記の項目を出力
   - 名前が格納されているアドレス
   - 名前

必須課題3-2:理解補助シート



malloc：「一つの構造体」を確保する例

```
typedef struct Person{
 char name[20]
 int age;
}Person;
```

```
Person *person;
person = ( Person *) malloc( sizeof( Person ) );
```

代入されるポインタ変数と同じ型にキャストする

この場合構造体１つ分を確保する（* 1を省略）

(型 *) malloc(**sizeof(型)** * **必要な個数**);

復習：構造体：構造体メンバへのアクセス

```
typedef struct Person{
 char name[20]
 int age;
}Person;
```

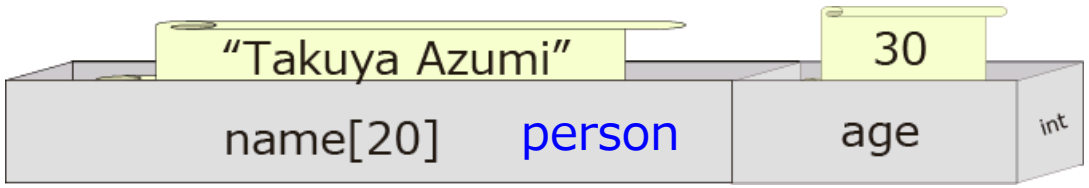
メンバのアクセス方法

構造体の変数名.メンバ名

```
Person person;

person.age = 30;
strcpy(person.name,” Takuya Azumi” );
```

strcpyは文字列のコピーを行う関数



構造体ポインタのメンバのアクセス方法

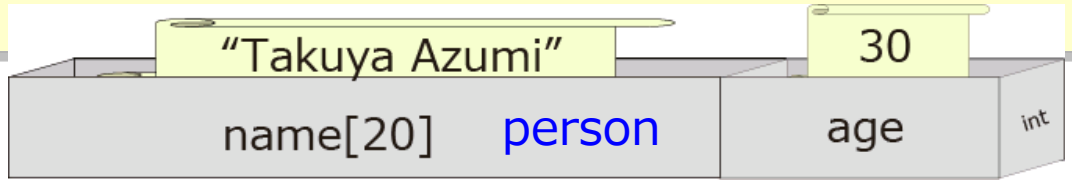
```
typedef struct Person{
 char name[20]
 int age;
}Person;
```

メンバのアクセス方法

ポインタ変数名->メンバ名

```
Person *person;

person = ( Person *) malloc( sizeof( Person ) );
//エラーチェック
person->age = 30;
strcpy(person->name,”Takuya Azumi”);
```



必須課題3-3:作成手順

1. 構造体のポインタ配列を定義
2. ファイルから読み込むデータ数（=行数）を取得
3. データが格納されているファイル名を取得
4. ２で入力されたデータ数分だけ、構造体のメモリ領域を動的メモリ確保（malloc）し、それぞれの先頭アドレスを１で定義したポインタ配列に確保
5. ３で指定されたファイルを開き、4で確保した構造体にデータを格納
6. 格納したデータを課題1-4と同様の形式でファイル（ファイル名は各自決める）に出力

著者リスト

1. 安積 卓也（情報システム学科）
2. 泉　朋子 （情報コミュニケーション学科）
3. 原田 史子（情報システム学科）