

プログラミング基礎

<http://bit.ly/prog2d>

動的なメモリ確保, 関数ポインタ

後期 第6週

2017/11/6


今回は

プログラムの実行中に**必要な分だけメモリ**
(つまり、データを格納する場所)
を確保する方法について説明します。

関数 `malloc()`, `free()` のためにインクルードする

- ・ `malloc()` … 実行時に（動的に）メモリを**確保する**関数
 - ・ `free()` … 実行時に（動的に）メモリを**解放する**関数
- (p.334～335)

```
1: #include <stdio.h>
2: #include <stdlib.h>
3:
4: int main(void)
5: {
6:     char *str;
7:     int num, i;
8:
9:     printf("num > ");
10:    scanf("%d", &num);
11:
```



メモリを確保して、ポインタstrでその場所を参照する
(この行の詳細は後のスライドで説明)

```
12:    str = (char *)malloc(sizeof(char)*(num+1));
13:    if(str==NULL) {
14:        printf("not allocated.\n");
15:        return 1;
16:    }
17:
```

メモリが確保できなかった場合、つまり、strが何も参照していない (NULL) の場合、プログラムを終了する。
(1を返しているのは、「異常終了」の意味)
(普段は「正常終了」の意味で0を返している)

確保したメモリに、文字を1文字ずつ格納する繰り返し処理
(numはscanfで入力した文字数)

```
18: for(i=0; i<num; i++) {  
19:     * (str+i) = 'a';  
20: }  
21: * (str+i) = '\0';  
22: printf("str: %s\n", str);  
23:  
24: free(str);  
25:  
26: return 0;  
27: }
```

strが参照しているi番目の
場所に文字「a」を代入

代入した文字の最後に終
端文字を代入する（つま
り、strが参照しているデー
タが文字列になる）

不要になったメモリを解放する
(ただし、プログラム終了時には、プログラムで使
用していた全てのメモリが自動的に解放される) (p.336)

mallocによるメモリ確保の詳細

1行のプログラムで以下の①～⑤を処理しています

① char型のサイズを得る (→1バイト)

② 何個分の確保したいのか計算する

⑤ 確保した先頭アドレスをstrに代入する (strで参照する)

```
str = (char *) malloc( sizeof(char) * (num+1) );
```

③ mallocは、引数に指定した分のメモリを確保する
(引数に指定した値の単位はバイト)

【例】 numが5の場合 「malloc(6)」 となり6バイト分確保される

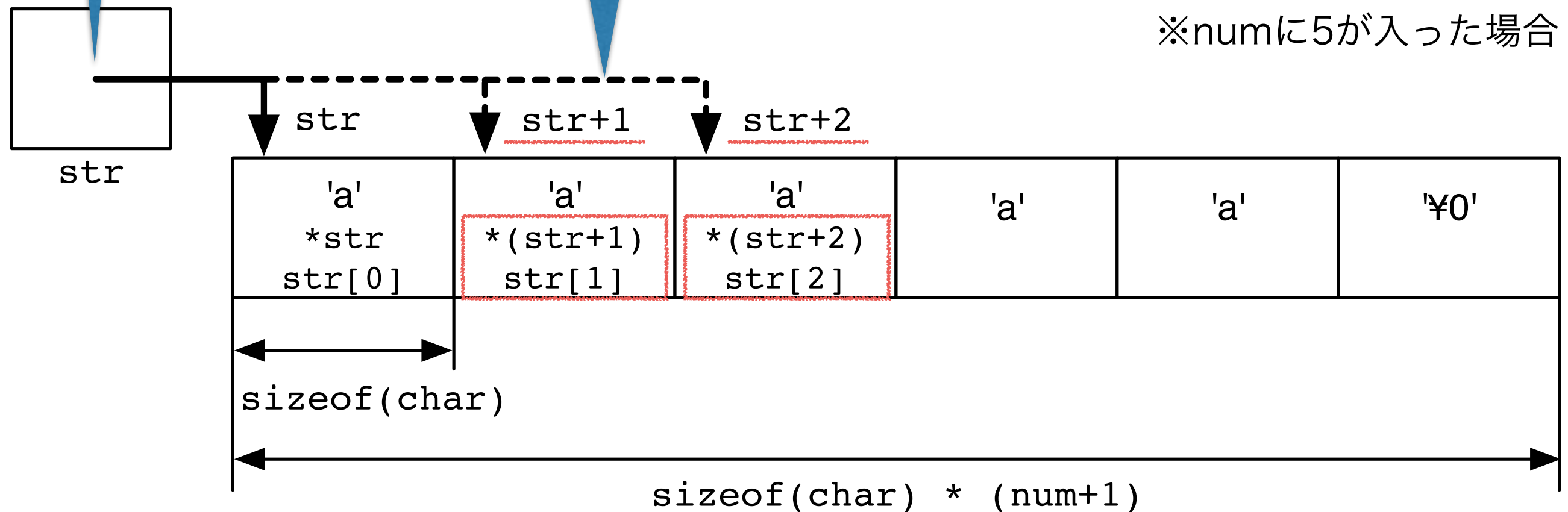
④ 確保した先頭アドレスの型を「(char *)」型にキャストする
(返される先頭アドレスの型はvoid (つまり型なし) のため)

メモリ確保のイメージ

確保されたメモリの先頭アドレス、つまり先頭文字を参照している

各要素への参照の仕方は配列で確保した時と同じ

※numに5が入った場合



今回はあともう一つ

プログラムで定義した関数を参照するための
ポインタである「**関数ポインタ**」
についても説明します。

関数ポインタとは (p.337)

関数で定義した処理内容（CPUへの命令の集合）も
メモリ上に格納されています



関数の定義が可能されているアドレスも
ポインタで参照することができます



関数を参照するためのポインタ 「関数ポインタ」



関数ポインタが参照する関数を変える
→呼び出す関数を切り替えることができる

```
1: #include <stdio.h>
2:
3: int max(int x, int y);
4: int min(int x, int y);
5:
6: int max(int x, int y)
7: {
8:     if(x > y) return x;
9:     else return y;
10: }
11:
12: int min(int x, int y)
13: {
14:     if(x < y) return x;
15:     else return y;
16: }
17:
```

引数の並びと戻り値の型が同じ
関数を2個定義する

- max … 大きい方を返す
- min … 小さい方を返す

関数ポインタを宣言する (構文はp.338参照)

```
18: int main(void)
19: {
20:     int (*pM)(int x, int y);
21:     int res;
22:
23:     pM = max;
24:     res = (*pM)(5, 10);
25:     printf("res: %d\n", res);
26:
27:     pM = min;
28:     res = (*pM)(5, 10);
29:     printf("res: %d\n", res);
30:
31:     return 0;
32: }
```

関数max()のアドレスを代入する

関数ポインタpMが参照
している関数を呼び出す
(構文はp.340参照)

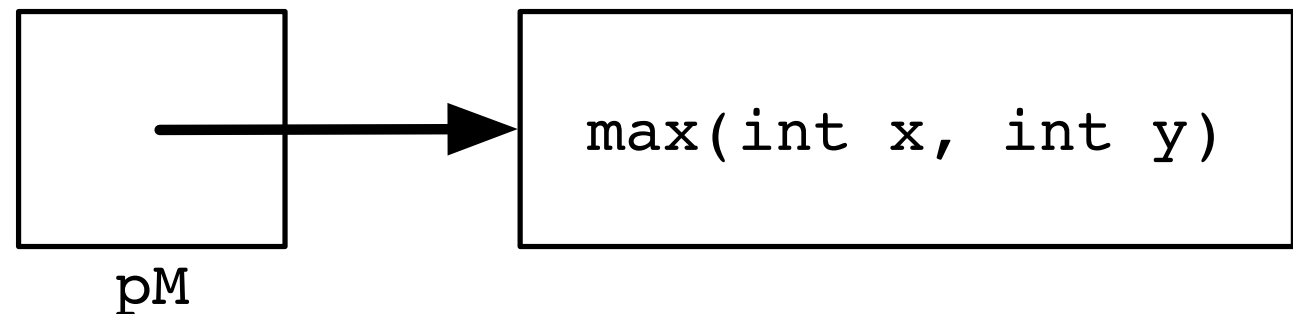
関数min()のアドレスを代入する

関数ポインタのイメージ

プログラムの記述

処理の様子

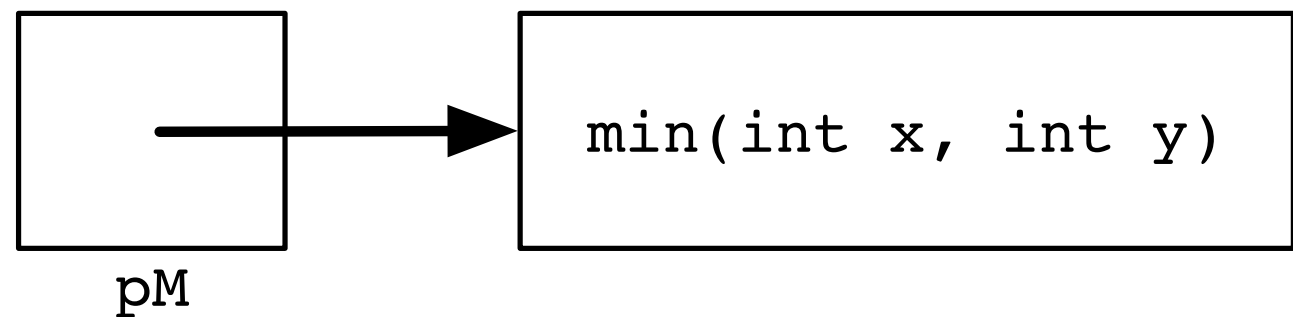
```
pM = max;
```



```
(*pM)(5, 10);
```

`max(5, 10)` が呼び出される

```
pM = min;
```



```
(*pM)(5, 10);
```

`min(5, 10)` が呼び出される

【練習6-1】

1つ目のサンプルプログラムを実行して、malloc()によるメモリ確保の動作を確認しましょう。

【練習6-2】

2つ目のサンプルプログラムを実行して、関数ポインタの動作を確認しましょう。

【課題6-1】

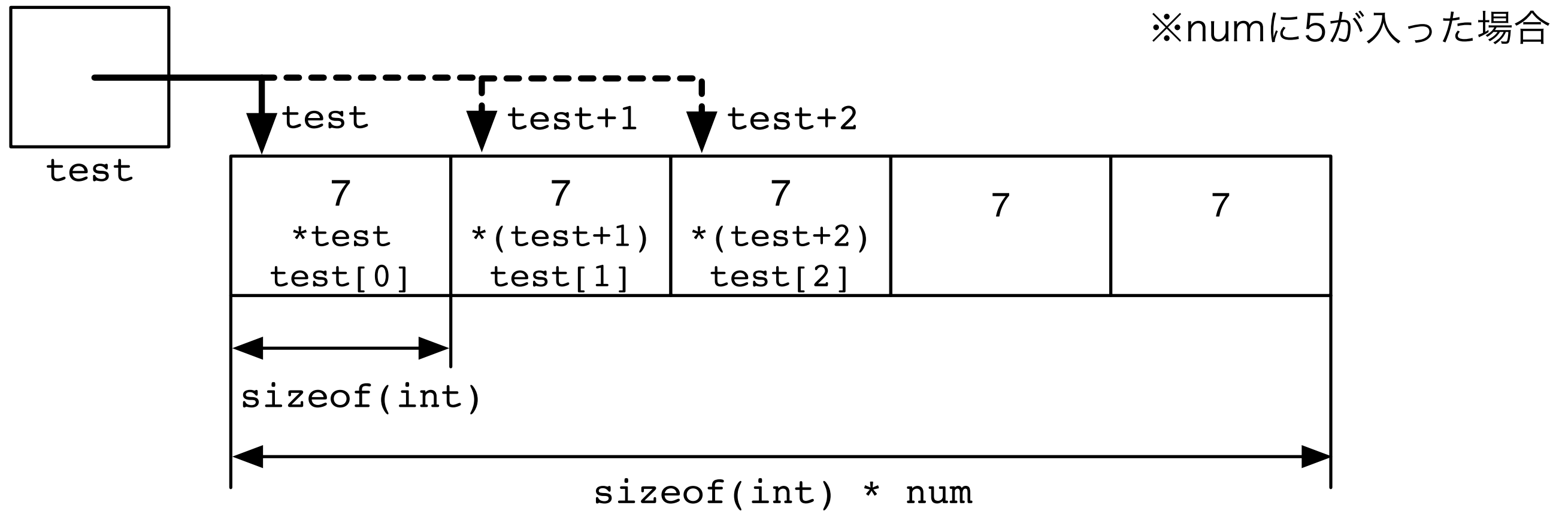
1つ目のサンプルプログラム（文字列つまりchar型の配列のメモリを確保する）を、int型の配列のメモリを確保するプログラムに変更してください。

（以下の補足を参考）

- ▶ int型のポインタを宣言する（例えばarray）
- ▶ malloc()の箇所をint型へ変更する（**終端文字の分は不要**）
array = (**int** *)malloc(sizeof(**int**)*num);
- ▶ 確保した配列に代入する値を整数へ変更する（例えば7）
- ▶ 配列の要素を**繰り返し出力する**処理を追加する

【課題6-1】

このプログラムのメモリ上のイメージ



【課題6-1】

[実行結果]

```
num > 5
```

```
7 7 7 7 7
```

【課題6-2】

1つ目のサンプルプログラムでの文字列を作る処理を、関数make_string()として作成してください。

[この関数のプロトタイプ宣言]

```
char *make_string();
```

```
/* サンプルプログラムで「numを入力する処理」～
```

```
「文字列に'\0'を代入する処理」までを関数に入れる */
```

```
/* 関数の最後の処理は、return str; とする
```

```
(確保したメモリのアドレスを返す) */
```

```
/* メモリ確保できなかった場合の return 1; は、
```

```
return NULL; とする */
```


【課題6-2】

[mainでの処理]

```
char *mystr;  
mystr = make_string();  
printf("mystr: %s\n", mystr);  
free(mystr);
```

[実行結果]

```
num > 7  
mystr: aaaaaaa
```

【課題6-3】

int型の配列のためのメモリを確保し、引数で指定した個数分の偶数が格納された配列を作る関数 `make_even()` を作成してください。

[この関数のプロトタイプ宣言]

```
int *make_even(int num);
```

```
/* 課題6-1で作成した処理を関数に入れる
```

```
    (関数の作り方は課題6-2を参照) */
```

```
/* 作成した配列には、0からnum個の偶数を格納する */
```

【課題6-3】

[mainでの処理]

```
array = make_even(7);  /* 0から7個の偶数が入った配列を作る */
for(i=0; i<7; i++) {
    printf("%d ", *(array+i));
}
printf("\n");
free(array);

array = make_even(10); /* 0から10個の偶数が入った配列を作る */
for(i=0; i<10; i++) {
    printf("%d ", *(array+i));
}
printf("\n");
free(array);
```

[実行結果]

0 2 4 6 8 10 12

0 2 4 6 8 10 12 14 16 18

【課題6-4】

char型の配列のためのメモリを確保し、引数で指定した個数分のアルファベットが格納された配列を作る関数fill_alpha()を作成してください。

```
char *fill_alpha(int num);
```

```
/* 課題6-2で作成した関数make_string()をベースに作成できる */
```

```
/* 作成した配列には、'a'からnum個のアルファベットを格納する */
```

【課題6-4】

[mainでの処理]

```
mystr = fill_alpha(5);  
printf("mystr: %s\n", mystr);  
free(mystr);
```

```
mystr = fill_alpha(20);  
printf("mystr: %s\n", mystr);  
free(mystr);
```

[実行結果]

```
mystr: abcde  
mystr: abcdefghijklmnopqrst
```


まだ余裕のある人は…

【課題6-5】

練習6-2で作成したプログラムに、**2つの引数の平均値を返す関数avr**を作成し、main()の関数ポインタpMを使って呼び出すプログラムを作成してください。

[この関数のプロトタイプ宣言]

```
int avr(int x, int y);
```

```
/* xとyの平均値を計算して返す */
```

```
/* 平均値はint型で計算する（小数点は切り捨てられる） */
```

[実行結果（関数ポインタpMに対して5と10を引数で渡した場合）]

```
res: 7
```

小テストの追試について

11月10日（金）試験終了後に、小テストの追試験を実施します。未受験の小テストがある人は、この時に必ずまとめて受けて下さい。

（やむを得ず欠席する際は事前に連絡を下さい。）

未受験分の小テストは0点として評価します。

今回実施した小テスト

- ▶ 第3週の内容
- ▶ 第4週の内容
- ▶ 第5週の内容

定期試験の実施について

試験中に使用できるもの

- 筆記用具

(メモ用紙が必要な人には試験中に配布する)

- 演習室のコンピューター台

(一つの机に一人の配置で、座る場所はどこでもよい)

定期試験の実施について

試験中に参照できるもの

- 自分のホームディレクトリ（ホームフォルダ）以下に
保存されているファイル
（定期試験では紙媒体のものは参照不可）
- 上記以外の情報を参照することは不正行為とする
例：USBで接続された機器に保存されているファイルの参照
ネットワークを介した情報の参照、など
- 試験中は、演習室外へのネットワークアクセスは遮断される