

# プログラミングII

<http://bit.ly/Prog3i>

## メモリの動的確保とポインタ

前期 第4週

2019/5/7

# 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

```
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)

# 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 ( $\rightarrow$  1バイト)

⑤ 確保した先頭アドレスをstr  
に代入する (strで参照する)

② 何個分の確保したいのか計算する

```
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)

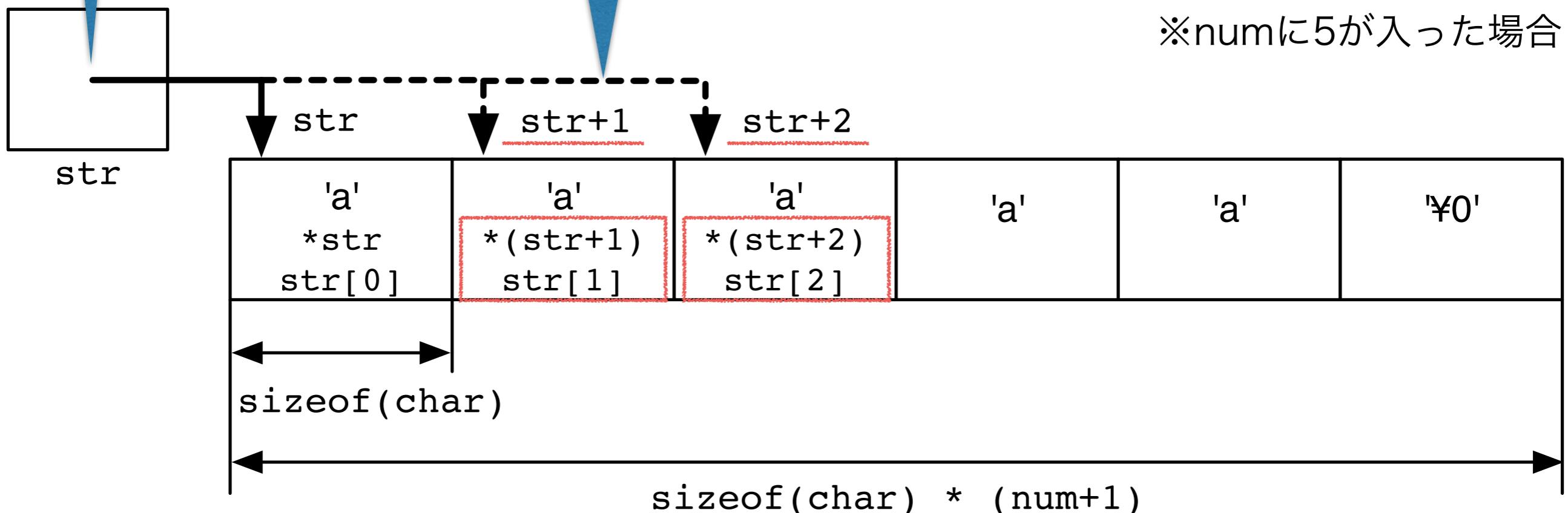
【例】 numが5の場合 「malloc(6)」 となり6バイト分確保される

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



# ポインタと配列について

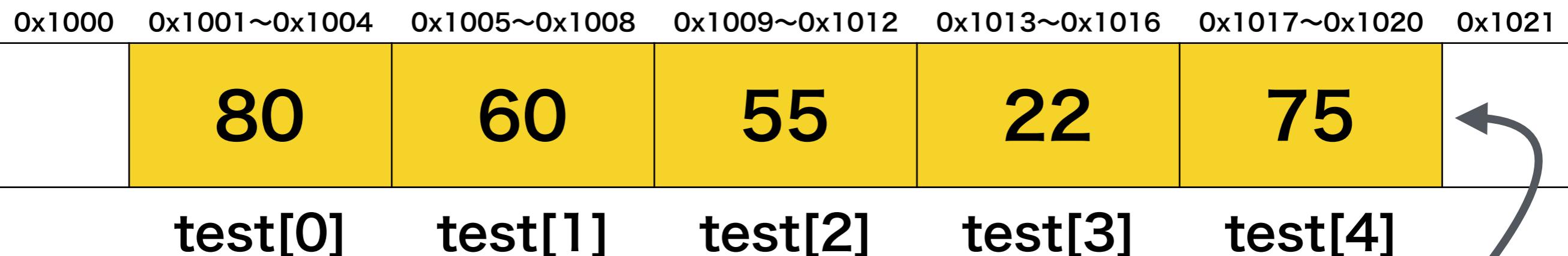
ポインタと配列については、以降に、D科2年生プログラミングでの資料を載せますので参考にしてください。（説明文中のページは、やさしいCのページ番号になります。）

# 配列とアドレスの関係

配列は、**同一のデータ型**、つまり同一のサイズの要素が、メモリ上に**連続して並んだ構造**をしています。

## 【例1】

```
int test[5] = {80, 60, 55, 22, 75};
```



メモリ上に連続して空いているアドレスに配列が確保される

# 配列のアドレスを出力する

配列testの各要素の値とアドレスを繰り返し出力する

【例2】

```
int i;
for(i=0; i<5; i++) {
    printf("test[%d]: %d, &test[%d]: %p\n",
           i, test[i], i, &test[i]);
}
```

配列testのi番目のアドレスを出力する (p. 301)

正確には、i番目の格納場所の先頭アドレス（下図赤枠）を出力する

	0x1000 ~ 0x1004	0x1005 ~ 0x1008	0x1009 ~ 0x1012	0x1013 ~ 0x1016	0x1017 ~ 0x1020	0x1021
	80	60	55	22	75	

test[0]    test[1]    test[2]    test[3]    test[4]

# 配列の先頭アドレス (p.304)

配列の**名前のみの表記**は、配列の**先頭要素のアドレス**、つまり0番目の要素のアドレスを表します。

## 【例3】

```
printf("test[0]: %d, &test[0]: %p\n",
       test[0], &test[0]);
printf("*test: %d, test: %p\n", *test, test);
```

配列testの先頭要素の値 (0番目の値)

配列testの先頭要素のアドレス (0番目のアドレス) (下図赤枠)

	0x1000	0x1001~0x1004	0x1005~0x1008	0x1009~0x1012	0x1013~0x1016	0x1017~0x1020	0x1021
		80	60	55	22	75	
	test[0]	test[1]	test[2]	test[3]	test[4]		

# ポインタの演算

ポインタは**加算/減算**の演算をすることができます。

(p.305 表10-1)

## 【例4】

```
int *ptr;  
ptr = &test[0];  
printf("test[1]: %d, &test[1]: %p\n",  
      test[1], &test[1]);  
printf("*ptr+1: %d, ptr+1: %p\n",  
      *(ptr+1), ptr+1);
```

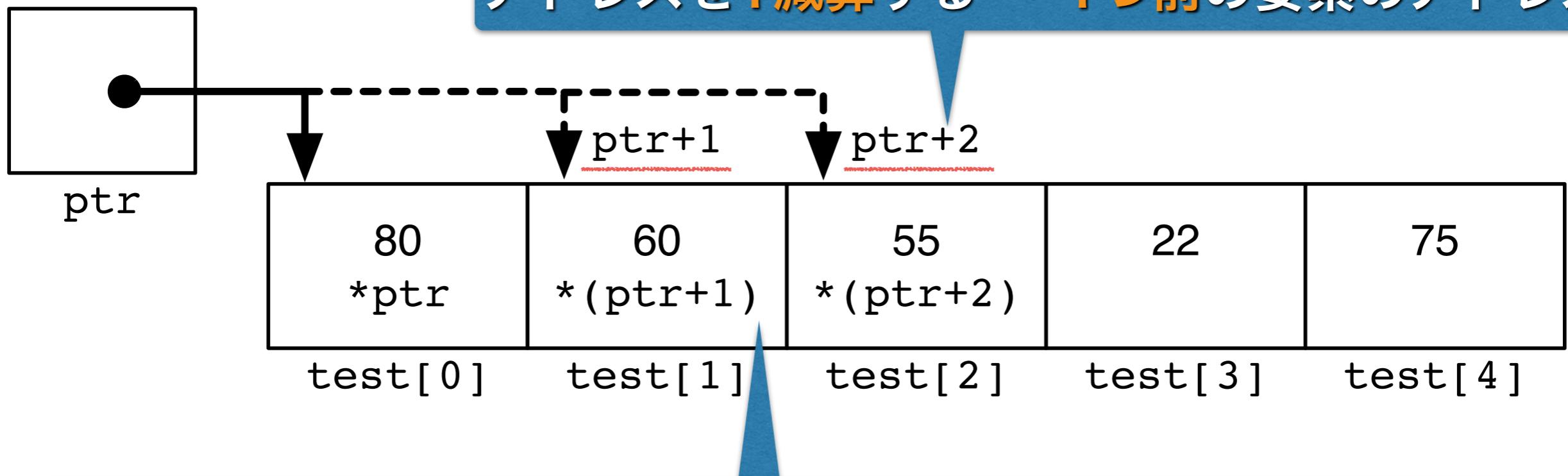
「ptrのアドレスに1を加算する」つまり「ptrが  
参照している要素の次の要素のアドレス」を意味する

「ptrが参照している要素の次の要素の値」

(先にアドレスを1加算して、間接参照演算をしていることに注目)

# ポインタの演算のイメージ

アドレスを1加算する → 1つ次の要素のアドレス  
アドレスを2加算する → 2つ次の要素のアドレス  
アドレスを1減算する → 1つ前の要素のアドレス



2つの演算「+」と「\*」が次の順で処理される

- ① ポインタの加算をする「`ptr + 1`」
- ② 間接参照演算でアドレスの参照先の値を得る「`*(ptr + 1)`」

# 繰り返し処理とポインタの演算

ポインタの演算を繰り返し処理を利用して、  
「ポインタが参照している配列に対する繰り返し処理」を作ります

## 【例5】

```
int *ptr;  
ptr=&test[0];  
for( i=0; i<5; i++ ) {  
    printf( "*ptr+%d: %d, ptr+%d: %p\n",  
           i, *(ptr+i), i, ptr+i );  
}
```

変数iを0から4まで変化させる

変数iをアドレスの加算に使うと、「ptrが参照  
している配列に対する繰り返し処理」が作れる

# 配列を引数として使う

## 【例6】

[関数の定義]

```
void show_range1(int t[], int s, int e)
```

```
{
```

```
    int i;
```

```
    for(i=s; i<=e; i++) {
```

```
        printf("t[%d]: %d, &t[%d]: %p\n",
               i, t[i], i, &t[i]);
```

```
}
```

```
}
```

[main()から呼び出す]

```
show_range1(test, 1, 3);
```

/\* show\_range1(&test[0], 1, 3); も同じ意味 \*/

仮引数に配列を使う際は、配列名  
の後に「[]」を付ける (p.309)

i番目の値とアドレスを表す

実引数には配列名（先頭  
要素のアドレス）を指定する

# ポインタを引数として使う

実際には配列の先頭要素のアドレスが参照渡しされて  
います。したがって、配列の仮引数をポインタとして  
も表します。 (p.311)

## 【例7】

```
void show_range2( int *ptr, int s, int e )
{
    int i;
    for( i=s; i<=e; i++ ) {
        printf( "*ptr+%d: %d, ptr+%d: %p\n",
                i, *(ptr+i), i, ptr+i );
    }
}
```

[main()から呼び出す]

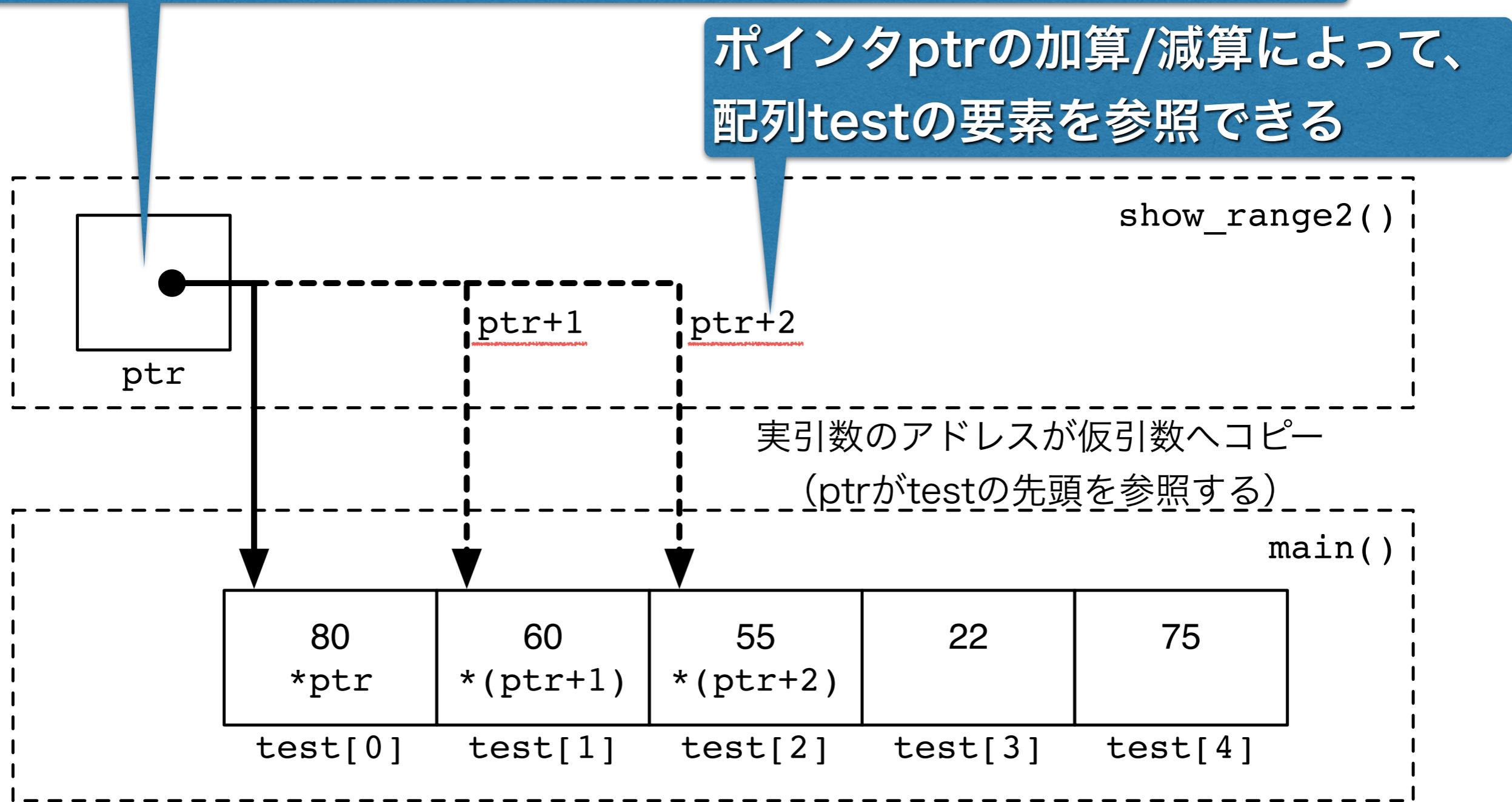
```
show_range2(test, 2, 4);
```

仮引数をポインタ表記にする

ポインタの演算で表す

# 配列の参照渡しのイメージ

参照渡しによって、ポインタptrが、main内で宣言されている配列testの先頭要素のアドレスを参照している (p.313)



# ポインタに添字演算子を使う

ポインタの演算を配列の添字演算子「[]」でも表すことができます。(p.314)

【例8】(例7を添字演算子で書き換えた)

```
void show_range2(int *ptr, int s, int e)
{
    int i;
    for(i=s; i<=e; i++) {
        printf("*ptr+%d: %d, ptr+%d: %p\n",
               i, ptr[i], i, &ptr[i]);
    }
}
```

添字演算子で次のように置き換えて、配列のように表記する

- 「\*(ptr+1)」 → 「ptr[i]」
- 「ptr+1」 → 「&ptr[i]」

# 【課題の準備】

演習室で作業する前に、以下のコマンドを入れるだけで準備が完了する

```
$ mygitclone 「自分のGitHubユーザ名」  
$ cd prog3i-(ユーザ名)  
$ ./myconf
```

※本体をシャットダウンするまでは、  
上記「mygitclone」と「myconf」の設定は有効です

# 【課題の準備】

以下の流れで、課題のプログラムを作るためのフォルダを準備しましょう。

1. 端末を起動して、以下のコマンドを実行して前期第4週のフォルダを作る

```
$ cd prog3i-(ユーザ名)          ←既に移動しているなら不要)
$ mkdir week104
$ cd week104
```

※課題で作るファイル名は各自で決めて構いません。  
(全ての課題を1つのファイルでまとめて作っても良い)

# 【練習4-1】

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。  
(配付資料と同じ場所からダウンロード可能)

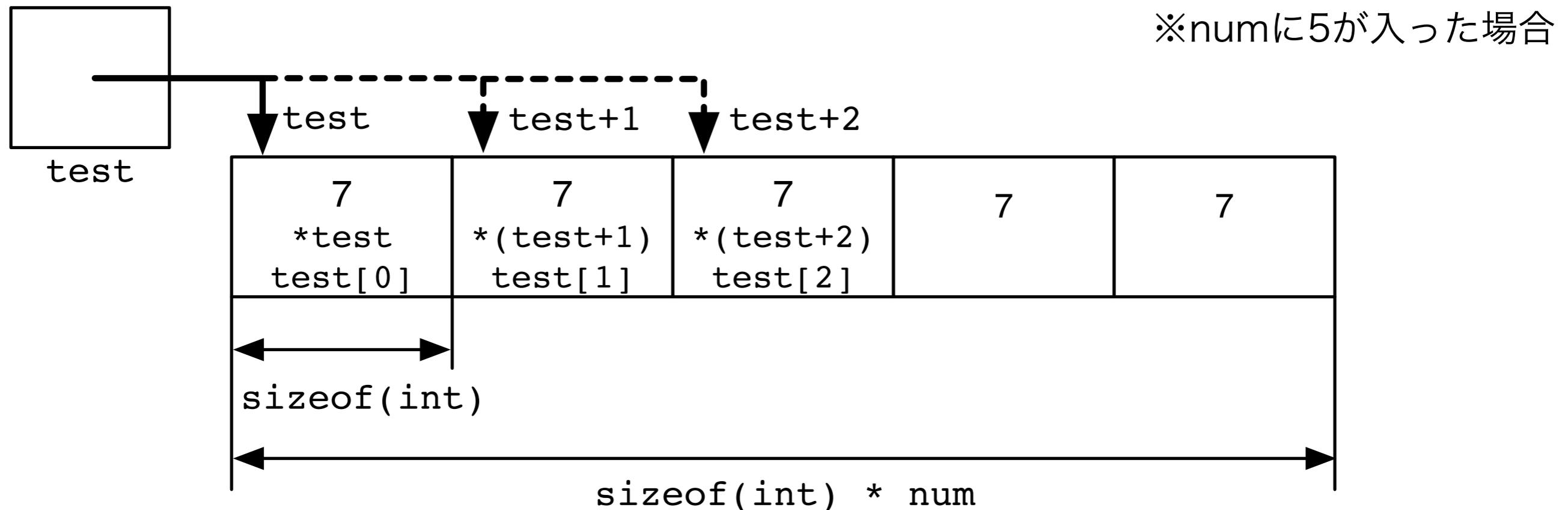
# 【課題4-1】

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（終端文字分の「+1」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4-1】

このプログラムのメモリ上のイメージ



# 【課題4-1】

[ 実行結果 ]

```
num > 5  
7 7 7 7 7
```

# 【課題の提出】

以下の流れで、作ったCプログラムをGitHubにプッシュして、Webサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A  
$ git commit -m "課題4-1提出"  
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

[https://github.com/nit-ibaraki-prog3i/prog3i-\(ユーザ名\)](https://github.com/nit-ibaraki-prog3i/prog3i-(ユーザ名))

# 【課題4-2】

サンプルプログラムの文字列を作る処理を、以下を参考に関数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4-2】

[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
```

# 【課題の提出】

以下の流れで、作ったCプログラムをGitHubにプッシュして、Webサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A  
$ git commit -m "課題4-2提出"  
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

[https://github.com/nit-ibaraki-prog3i/prog3i-\(ユーザ名\)](https://github.com/nit-ibaraki-prog3i/prog3i-(ユーザ名))

# 【課題4-3】

**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4-3】

[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
```

# 【課題の提出】

以下の流れで、作ったCプログラムをGitHubにプッシュして、Webサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A  
$ git commit -m "課題4-3提出"  
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

[https://github.com/nit-ibaraki-prog3i/prog3i-\(ユーザ名\)](https://github.com/nit-ibaraki-prog3i/prog3i-(ユーザ名))

# 【課題4-4】

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4-4】

[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
```

# 【課題の提出】

以下の流れで、作ったCプログラムをGitHubにプッシュして、Webサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A  
$ git commit -m "課題4-4提出"  
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

[https://github.com/nit-ibaraki-prog3i/prog3i-\(ユーザ名\)](https://github.com/nit-ibaraki-prog3i/prog3i-(ユーザ名))