プログラミング基礎

http://bit.ly/prog2d

配列とポインタ

後期 第4週

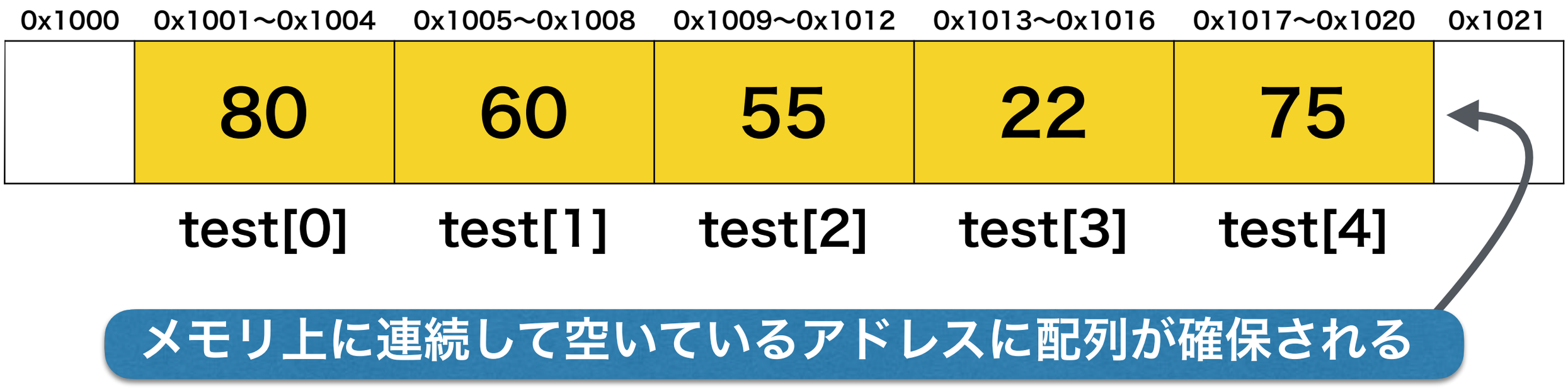
2017/10/16

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};
```



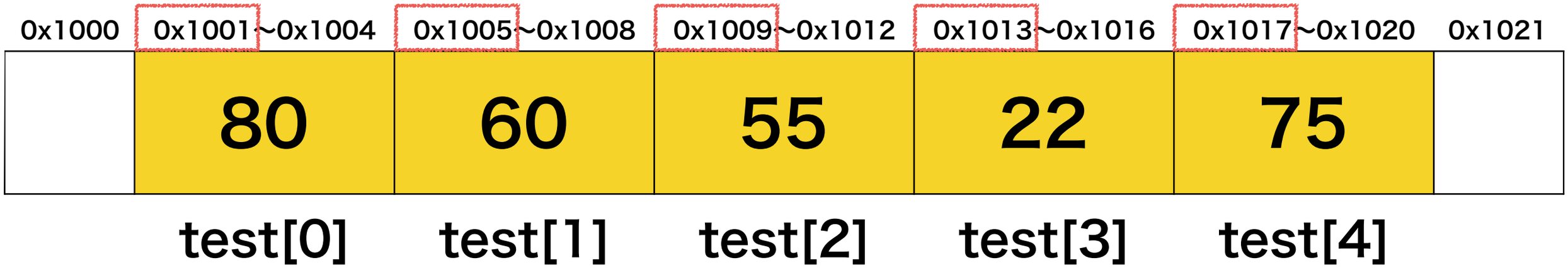
配列のアドレスを出力する

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



配列の先頭アドレス（p.304）

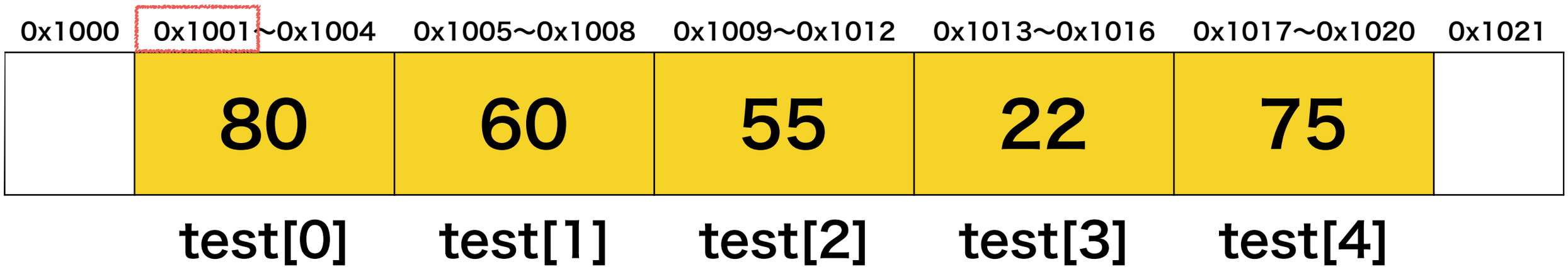
配列の名前のみの表記は、配列の先頭要のアドレス、
つまり0番目の要素のアドレスを表します。

【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）



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】

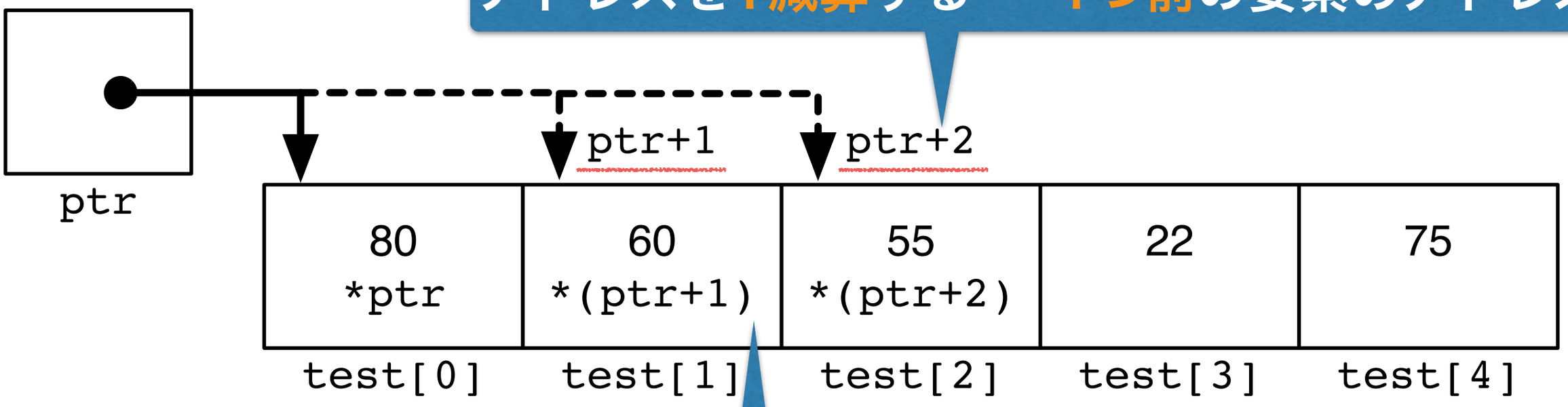
```
int *ptr;
ptr = &test[0];
printf("test[1]: %d, &test[1]: %p\n”,
        test[1], &test[1]);
printf("*ptr+1: %d, ptr+1: %p\n”,
        *(ptr+1), ptr+1);
```

「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「prt+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に利用して、
「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】

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

[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 */
```

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】

仮引数をポインタ表記にする

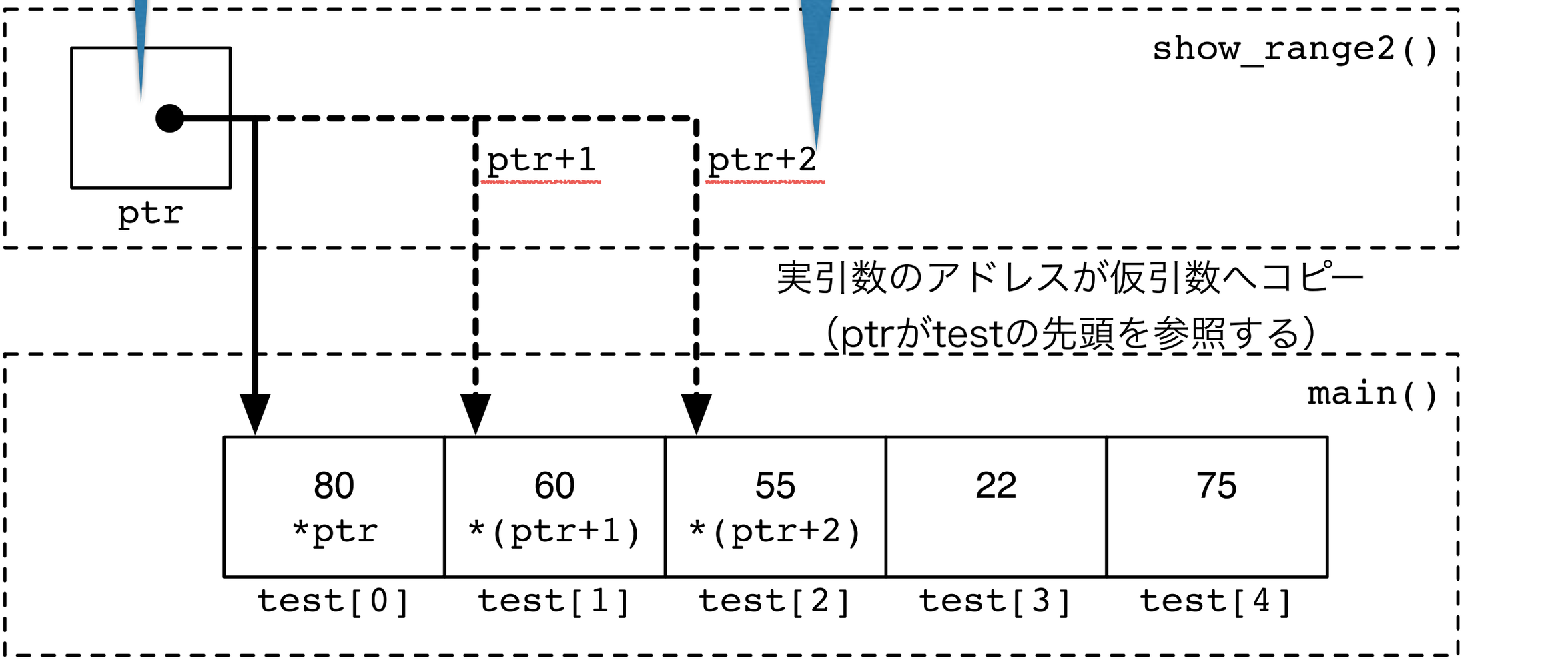
```
void show_range2(int *ptr, int s, int e)
{
    int i;
    for(i=s; i<=e; i++) {
        printf("*ptr+%d: %d, ptr+%d: %p\n”,
                i, *(ptr+i), i, ptr+i);
    }
}
[main()から呼び出す]

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

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ptrの加算/減算によって、配列testの要素を参照できる



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]」

話をまとめると・・・

前期の授業では、配列を扱う関数を作る場合、

配列はグローバル変数として宣言していた

↓

その配列しか処理できない関数

(処理対象の配列が固定された関数)

↓

引数に配列を使うことにより、

関数で処理できる配列が固定されなくなる

↓

より汎用的な関数を作ることができるようになる

【練習4-1】

例1と例2にある処理をmain()内に書いて、プログラムを実行し、配列の各要素のアドレスを出力してみましょう。

【練習4-2】

練習4-1で作ったmain()内に、続けて例5の処理を書いて、プログラムを実行し、ポインタによる配列の繰り返し処理の結果を出力してみましょう。

【練習4-3】

練習4-2で作ったプログラムに、例7で示した関数show_range2()の定義を追加し、main()内の続きにこの関数の呼び出しを追加して、プログラムを実行し、関数への参照渡しの処理を確認してみましょう。

【課題4-1】

仮引数のポインタが参照する配列に対して、指定された範囲内の要素の合計を求める関数sum_range()を作成してください。

[この関数のプロトタイプ宣言]

```
int sum_range(int *ptr, int s, int e);

  /* ptrが参照する配列のs番目からe番目までの要素の合計を求め、
        結果をreturnで戻す */
  /* show_range2()を参考に作れる */
```

【課題4-1】

[mainでの処理]

```
 /* 配列はグローバルに宣言するのではなく、main()内で宣言する */

 int test[5]  = {80, 60, 55, 22, 75};
 int test2[5] = {76, 85, 47, 92, 68};
 printf("sum_range: %d\n", sum_range(test, 1, 3));
 printf("sum_range: %d\n", sum_range(test2, 2, 4));
```

[実行結果]

```
 sum_range: 137
 sum_range: 207
```

【課題4-2】

仮引数のポインタが参照する配列に対して、指定された範囲内の最大値を求める関数find_max()を作成してください。

[この関数のプロトタイプ宣言]

```
int find_max(int *ptr, int s, int e);

  /* ptrが参照する配列のs番目からe番目までの要素の最大値を探し、
        結果をreturnで戻す */
```

【課題4-2】

[mainでの処理]

```
 /* 配列はグローバルに宣言するのではなく、main()内で宣言する */

 int test[5]  = {80, 60, 55, 22, 75};
 int test2[5] = {76, 85, 47, 92, 68};
 printf("find_max: %d\n", find_max(test, 0, 4));
 printf("find_max: %d\n", find_max(test2, 0, 2));
```

[実行結果]

```
 find_max: 80
 find_max: 85
```

【課題4-3】

仮引数のポインタが参照する配列に対して、偶数かどうかを判定し、その結果を配列に反映する関数check_even()を作成してください。

[この関数のプロトタイプ宣言]

```
void check_even(int *ptr);

  /* ptrが参照する配列の0番目から4番目までの各要素が偶数かを調べる  */
  /* 偶数の場合、その要素の値は残り、
        偶数でない場合、その要素の値は0となる  */
```

【課題4-3】

[mainでの処理]

```
 int test[5]  = {80, 60, 55, 22, 75};
 int test2[5] = {76, 85, 47, 92, 68};
 check_even(test);
 show_range2(test, 0, 4);
 check_even(test2);
 show_range2(test2, 0, 4);
```

[実行結果]

```
 *ptr+0: 80, ptr+0: 0x7fff5097e920     (←配列testの出力)
 *ptr+1: 60, ptr+1: 0x7fff5097e924
 *ptr+2: 0, ptr+2: 0x7fff5097e928
 *ptr+3: 22, ptr+3: 0x7fff5097e92c
 *ptr+4: 0, ptr+4: 0x7fff5097e930
 *ptr+0: 76, ptr+0: 0x7fff5097e900     (←配列test2の出力)
 *ptr+1: 0, ptr+1: 0x7fff5097e904
 *ptr+2: 0, ptr+2: 0x7fff5097e908
 *ptr+3: 92, ptr+3: 0x7fff5097e90c
 *ptr+4: 68, ptr+4: 0x7fff5097e910
```

【課題4-4】

仮引数のポインタptr1とptr2が参照する2つの配列に対して、それぞれの配列の同じ添字の要素の大きさを比較し、その結果をptr1が参照する配列に反映する関数check_bigger()を作成してください。

[この関数のプロトタイプ宣言]

```
void check_bigger(int *ptr1, int *ptr2);

  /* ptr1とptr2が参照する配列の0番目から4番目までの
        要素の大きさを比較する */
  /* 例えば、ptr1の0番目とptr2の0番目の要素を比較した場合…、*/
  /* ptr1の方が大きい場合は1、ptr2の方が大きい場合は2の値が、
        ptr1の0番目の要素に代入される */
```

【課題4-4】

[mainでの処理]

```
 int a1[5] = {8, 4, 2, 9, 0};
 int a2[5] = {5, 6, 1, 3, 7};
 check_bigger(a1, a2);
 show_range2(a1, 0, 4);
```

[実行結果]

```
 *ptr+0: 1, ptr+0: 0x7fff5097e8c0
 *ptr+1: 2, ptr+1: 0x7fff5097e8c4
 *ptr+2: 1, ptr+2: 0x7fff5097e8c8
 *ptr+3: 1, ptr+3: 0x7fff5097e8cc
 *ptr+4: 2, ptr+4: 0x7fff5097e8d0
```

まだ余裕のある人は…

【課題4-5】

**仮引数のポインタptr1が参照する配列に対して、
要素が逆順となった配列をptr2の参照先に作る関数
reverse()を作成してください。**

[この関数のプロトタイプ宣言]

```
void reverse(int *ptr1, int *ptr2);

  /*  「ptr1が参照する配列の0番目から4番目までの要素を、
        ptr2が参照する配列の後ろから代入する」という処理を繰り返す  */
```

【課題4-5】

[mainでの処理]

```
int a2[5] = {5, 6, 1, 3, 7};
int a3[5] = {1, 4, 2, 5, 3};
int ans[5] = {0, 0, 0, 0, 0};
reverse(a2, ans);
show_range2(ans, 0, 4);
reverse(a3, ans);
show_range2(ans, 0, 4);
```

[実行結果]

```
*ptr+0: 7, ptr+0: 0x7fff5097e8e0      (←配列a2の逆順)
*ptr+1: 3, ptr+1: 0x7fff5097e8e4
*ptr+2: 1, ptr+2: 0x7fff5097e8e8
*ptr+3: 6, ptr+3: 0x7fff5097e8ec
*ptr+4: 5, ptr+4: 0x7fff5097e8f0
*ptr+0: 3, ptr+0: 0x7fff5097e8e0      (←配列a3の逆順)
*ptr+1: 5, ptr+1: 0x7fff5097e8e4
*ptr+2: 2, ptr+2: 0x7fff5097e8e8
*ptr+3: 4, ptr+3: 0x7fff5097e8ec
*ptr+4: 1, ptr+4: 0x7fff5097e8f0
```

小テストについて

小テストの注意点

- □ 他人の力は借りずに、自分だけでプログラムを作成する。（つまり定期試験と同様）

- □ 小テスト中は、演習室外へのネットワークアクセスは遮断される。

小テストについて

小テスト中に参照できるもの

□ **教科書, 配付資料**

□ **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**

□ **小テストでは紙媒体のものは参照可能**

□ **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