C Programming

— Pointers—and—Arrays—

Outline

- Type of Pointers
- Calculation of Pointers
- Array and Pointers
- Exercise

Type of Pointers

- A pointer is a variable that contains the address of a variable

- A pointer can be declared as int or double type

- But the pointer is not independent, we call「the pointer to int type」or「the pointer to double type」.

  Ex int *p_a;
  double *p_b;

Type of Pointers

- Why we need to make the difference?

⇒ even we know the address, the memory allocated to store the data can not be determined.

Because int and double has different memory allocation

- An int (four bytes) will be 4-byte aligned
- A double (eight bytes) will be 8-byte aligned

.

(Reference)

- 1Kbyte== 1024byte
- 1Mbyte== 1024Kbyte 1024 == 2^{10}
- 1Gbyte== 1024Mbyte

Calculation of Pointers

Pointers can be calculated as follows

- add,subtraction
  p_a + 1
  p_b + 2

- increment,decrement
  p_a++
  p_b--

- subtraction

Calculation of Pointers

pointer4.c

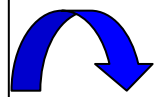
```
#include <stdio.h>

int main(void)
{
    int a;
    int *p_a;
    double b;
    double *p_b;

    p_a = &a;
    printf("p_a ... %p¥n", p_a);
    p_a++;
    printf("p_a ... %p¥n", p_a);

    p_b = &b;
    printf("p_b ... %p¥n", p_b);
    p_b++;
    printf("p_b ... %p¥n", p_b);

    return 0;
}
```



Calculation of Pointers

「pointer4.c」：

p_a++;　← add 1 to p_a(int)

p_b++;　← add 1 to p_b(double)

「pointer4.c」' result....

```
$ ./pointer4 ↵
p_a ... 0xbffff6a4
p_a ... 0xbffff6a8   +4
p_b ... 0xbffff698
p_b ... 0xbffff6a0   +8
```

the address of pointers,

「 increment by 4 for int poniter,
increment by 8 for double poniter」

⇒ the increment-size depends on the type of pointers

Array and Pointers

array1.c

```
#include <stdio.h>

int main(void)
{
  int array[5];
  int i;

  for (i=0; i<5; i++)  array[i] = i + 10;
  for (i=0; i<5; i++)  printf("%d¥n", array[i]);
  for (i=0; i<5; i++)  printf("&array[%d] ... %p¥n", i, &array[i]);

  return 0;
}
```

Array and Pointers

「array1.c」's result....

```
$ ./array1 ↵
  ...
&array[0] ... 0xbffff680
&array[1] ... 0xbffff684
&array[2] ... 0xbffff688
&array[3] ... 0xbffff68c
&array[4] ... 0xbffff690
```

- the address of each element is 4 byte away.
- if we add 1 to the pointer, its address will changed according to the type of pointer.

⇒ use pointers to show the elements of an array.

Array and Pointers

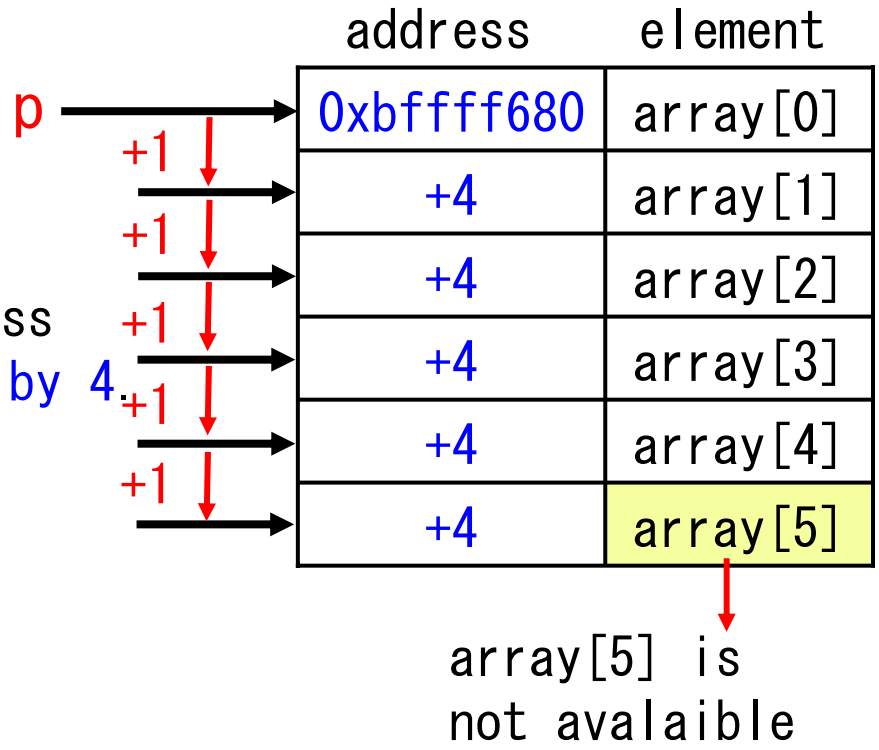
use pointers to show the elements of an array.

- because array's type is int, declare pointer to int type :
  int *p;

- when we increase p by 1, the addess which p points will be increased by 4.

- At the first step,p pointers to the first element (array[0]) :
  p = &array[0]

- increase p until that p points to the last element



Array and Pointers

array2.c

```
#include <stdio.h>

int main(void)
{
  int array[5];
  int *p;
  int i;

  for (i = 0; i < 5; i++)  array[i] = i + 10;
  for (p = &array[0]; p != &array[5]; p++)  printf("%d¥n", *p);

  return 0;
}
```

Array and Pointers

the for loop in 「array2.c」 can be revised

```
for (p = &array[0]; p != &array[5]; p++)
                                    printf("%d¥n", *p);
```

```
p = &array[0];
for (i = 0; i < 5; i++)  printf(%d¥n, *(p + i));
```

Array and Pointers

```
p = &array[0];
for (i = 0; i < 5; i++)  printf(%d¥n, *(p + i));
```

*(p + i)

by using [] (subscript operator), it can be written

which means,

p[i] ⇔ *(p + i)

For example, p[3] = *(p + 3)

Array and Pointers

- if we declare
  int array[5];

⇒ array becomes the pointer to its first element.

⇒ array[i] = *(array + i)

```
p = &array[0];
for (i = 0; i < 5; i++)  printf(%d¥n, *(p + i));
```

```
for (i = 0; i < 5; i++)  printf(%d¥n, *(array + i));
```

Exercise

revise「my_sort6.c」as follows

1. define「swap()function」to exchange values,

2. use pointers to process all elements in score[]