Basis and Practice in Programming

week 8

Array

- Character string constant
  - Constant property
  - Never be changed

```
printf("Hello World!₩n");
```

- Character string variable
  - Variable property
  - Can be changed

```
char str1[5]="Good";
char str2[]="morning";
```

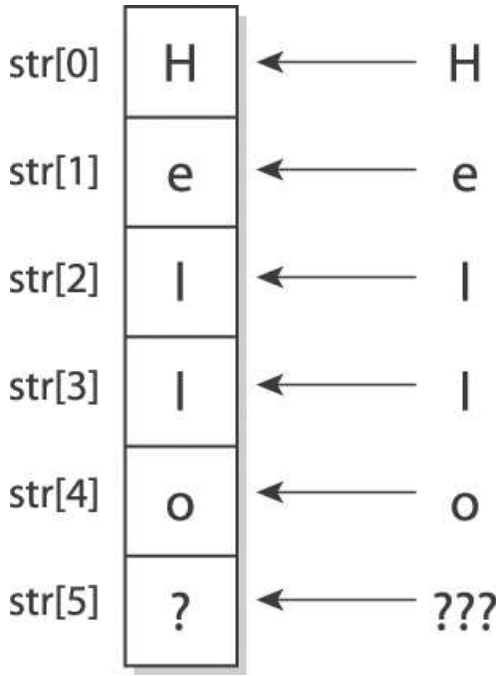
Array

```
/* Week 8 example 1 */

#include <stdio.h>

int main(void)
{
    char str1[5]="Good";
    char str2[]="morning";

    printf("%s₩n", str1);
    printf("%s %s₩n ", str1, str2);

    return 0;
}
```

Array

- Characteristic of character string
  - String ends with ‘null’ character
  - Null character is represented as ‘₩0’

```
int main(void)
{
    char str[6]="Hello";
    printf("Hello");
    . . . . .
```



Array

- The reason why null character is needed
  - It means the end of character string
  - A role of barrier between garbage values and real character string area
  - Printf function determines output length through the null character

```
int main(void)
{
    char str[100]=”Hello World!”;
    printf(“%s ₩n”, str);
    . . . . .
```

Array

```
/* Weed 8 example 2 */
  #include <stdio.h>

  int main(void)
  {
        int i;
        char ch;
        char str[6]="Hello";

        printf("Character string before modification₩n");
        printf("%s ₩n", str);

        for(i=0; i<6; i++)
              printf("%c | ", str[i]);

        /* modyfying character string */
        for(i=0; i<3; i++)
        {
              ch=str[4-i];
              str[4-i]=str[i];
              str[i]=ch;
        }

         printf("₩n₩nCharacter string after modification₩n");
         printf("%s ₩n", str);
         return 0;
}
```

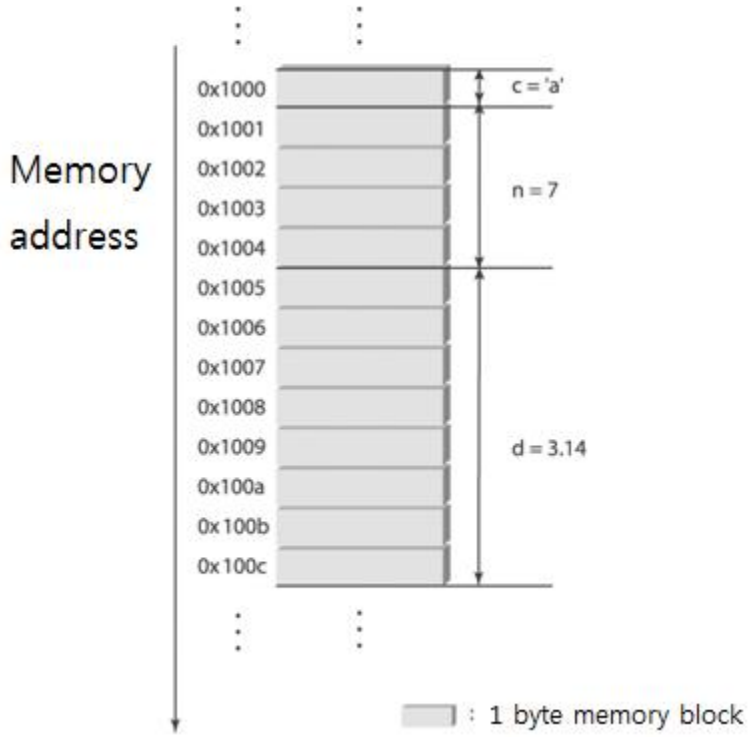
Array

```
/* Weed 8 example 3 */
#include <stdio.h>

int main(void)
{
    char str[30];

    printf("Please input character string : ");
    scanf("%s", str);

    printf("Character string from user: %s ₩n", str);

    return 0;
}
```

Pointer basic

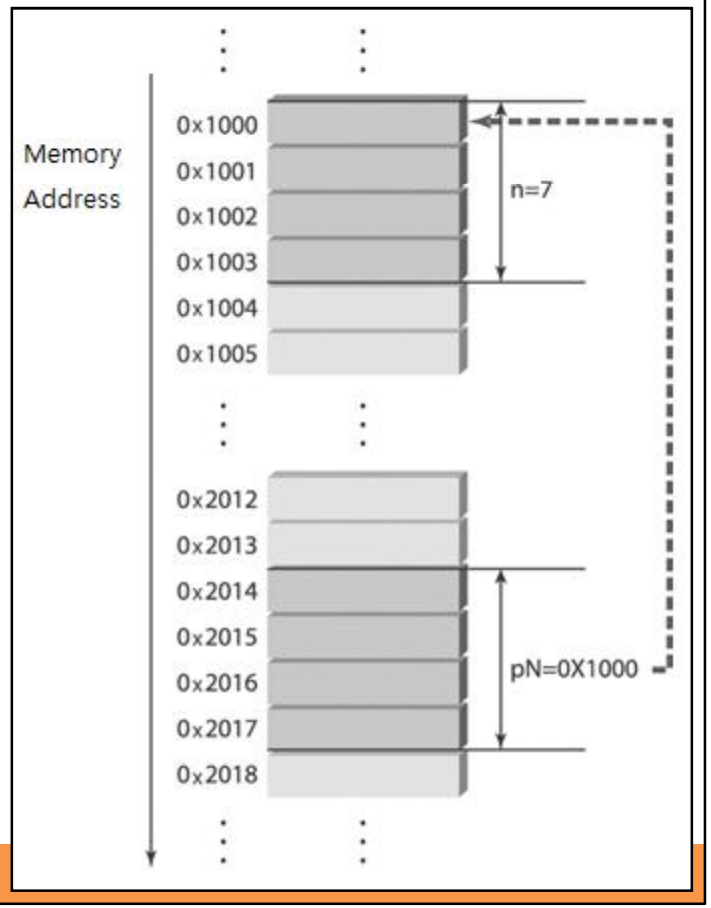
- Pointer variable
  - A variable to hold memory address
  - Pointer variable is so-called pointer

```
int main(void)
{
    char c='a';
    int n=7;
    double d=3.14;
    . . . . .
```



Pointer basic

- Pointer variable
  - The size of pointer depends on addressing system on computers
  - 32 bit computer: 4 bytes
  - Pointer pN points the variable n
    - Variable n is stored on 0x1000
    - Pointer pN hold the value 0x1000



Pointer basic

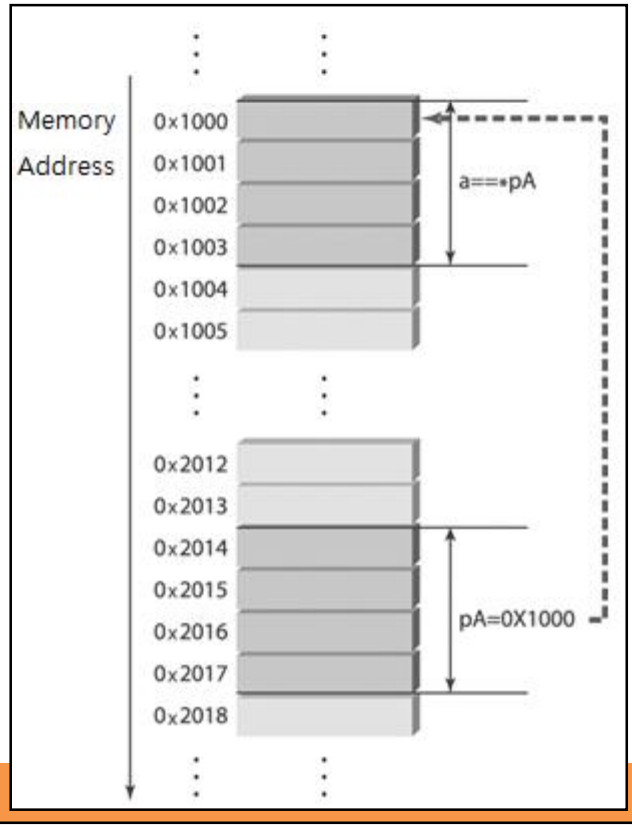
- Types and declaration of pointer variable
  - Declared with operator '*'
  - The type of pointer
    - It represents what type of data the pointer variable points

```
int main(void)
{
   int *a;              // named as 'a' and can points data type of 'int'
   char *b;             // named as 'b' and can points data type of 'char'
   double *c;           // named as 'c' and can points data type of 'double'
. . . . .
```

Pointer basic

- Pointer operators
  - & operator : 'address of'
    - Returns the address of a variable
  - * operator : 'refers the address'
    - Refers memory address which the pointer points

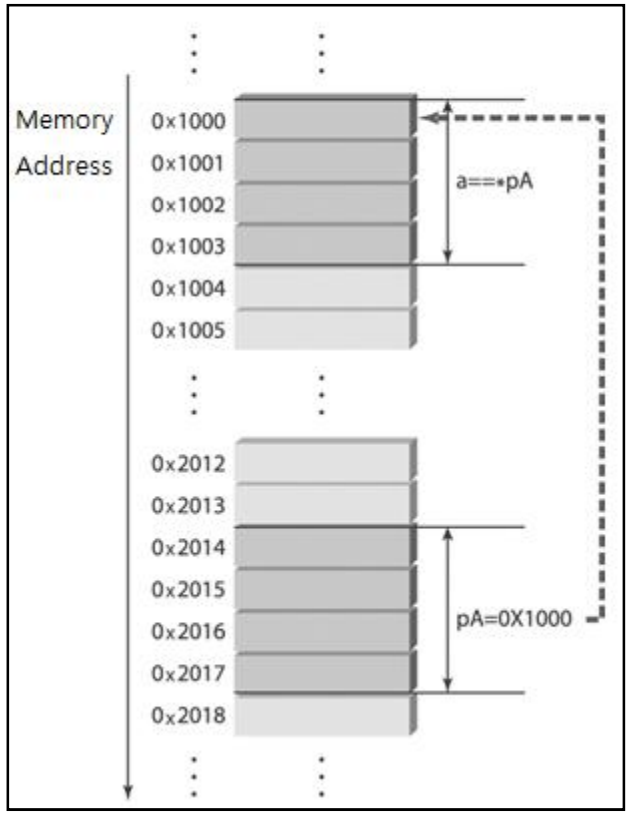
```
int main(void)
{
   int a=2005;
   int *pA=&a;
   printf("%d", a);       //direct access
   printf("%d", *pA);  // indirect access
   . . . . .
```



Pointer basic

- Pointer operators

```
/* Week 8 example 4 */
#include <stdio.h>

int main(void)
{
    int a=2005;
    int* pA=&a;

    printf("pA : %d ₩n", pA);
    printf("&a : %d ₩n", &a);

    (*pA)++;            // the same meaning as 'a++'

    printf("a   : %d ₩n", a);
    printf("*pA : %d ₩n", *pA);

    return 0;
}
```



Pointer basic

- Why diverse types of pointer exist?
  - Pointer type determines how much amount of memory to refer

```
#include <stdio.h>
 int main(void)
{
    int a=10;
    int *pA = &a;
    double e=3.14;
    double *pE=&e;

    printf("%d %f", *pA, *pE);
    return 0;
}
```

Pointer basic

- Example 1

```
int main(void)
{
   int *pA;       // pA is initialized with garbage value
   *pA=10;
   return 0;
}
```

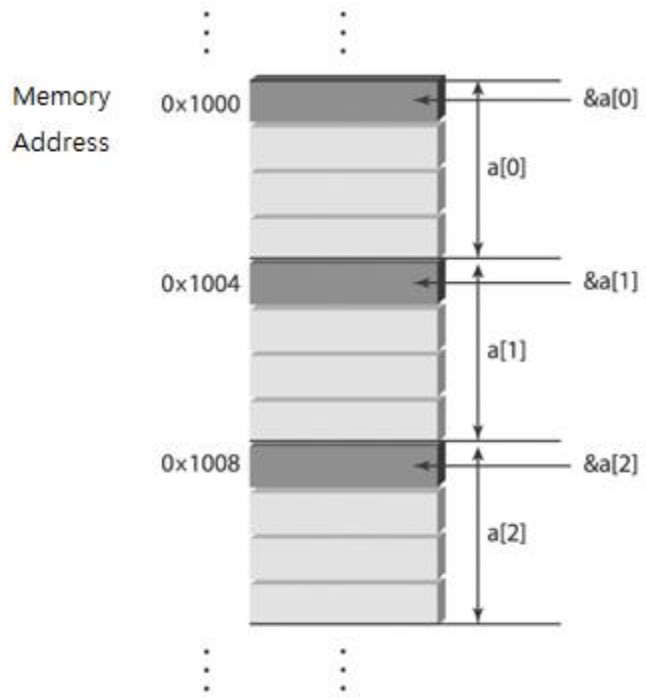
- Example 2

```
int main(void)
{
   int* pA=100;   // What the hell where the address 100 is?
   *pA=10;
   return 0;
 }
```

Relation between Pointer and Array

- The name of array
  - The name of a array represents the address of 1st element of the array

```
int a[5]={0, 1, 2, 3, 4}
```



Relation between Pointer and Array

```
/* Week 8 example 5 */

#include <stdio.h>

int main(void)
{
    int a[5]={0, 1, 2, 3, 4};

    printf("%d, %d ₩n", a[0], a[1]);
    printf("Address of a[0] : %d, address of a[1] : %d₩n", &a[0], &a[1]);
    printf("Name of array : %d ₩n", a);

    return 0;
}
```

Relation between Pointer and Array

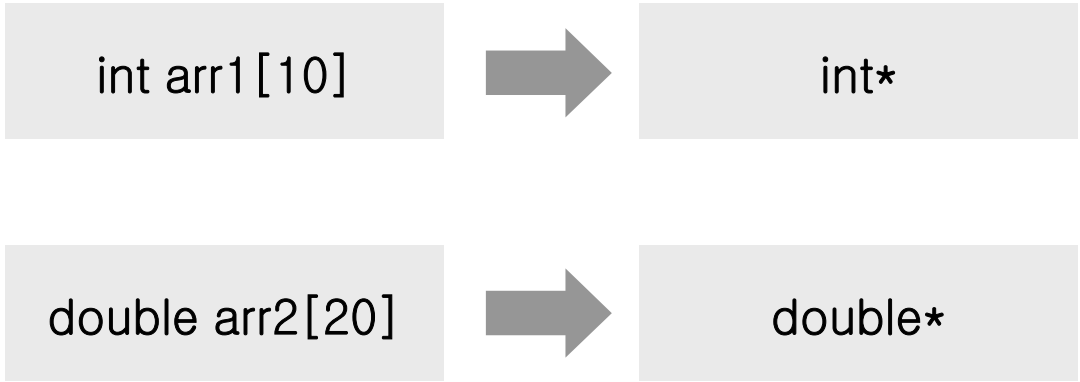
- Comparison between pointer and array

	Pointer	**Name of array**
Name exists?	Yes	**Yes**
What represents?	Address of memory	**Address of memory**
Variable or constant?	Variable	**Constant**

```
int main(void)
{
    int a[5]={0, 1, 2, 3, 4};
    int b=10;
    a=&b;   // an error occurs! The array a is constant
            // if the array a was a pointer variable, it would be OK
}
```

Relation between Pointer and Array

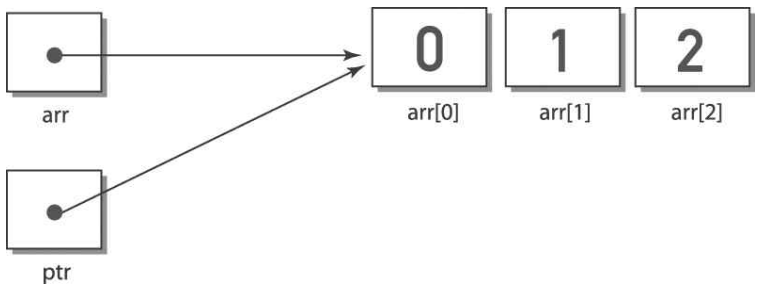
- The data type of array name
  - Since array name is also a pointer, it has data type too



Relation between Pointer and Array

- Utilization of array name
  - An array name as a pointer, pointer as a array name

```
/* Week 8 example 6 */
#include <stdio.h>

int main(void)
{
    int arr[3]={0, 1, 2};
    int *ptr;

    ptr=arr;

    printf("%d, %d, %d ₩n", ptr[0], ptr[1], ptr[2]);
    return 0;
}
```



Exercise

- Find palindrome (Due date : Today's 11:59 PM)
  - Get one string as input
  - Print "yes" if input string is a palindrome
  - Print "no" if input string is not a palindrome
  - Length of input string <= 100
  - Hint) Implements "strlen" function that counts the number of characters in string (using null char)