C 프로그래밍 프로젝트

Chap 13. 포인터와 배열! 함께 이해하기

2013.10.02.

오 병 우

컴퓨터공학과

kit 국립금오공과대학교
Kumoh National Institute of Technology

Programming in C,
정재은 저,
사이텍미디어. 9장
참조

(교재의 13-1은
읽지 말 것)

13-1 포인터와 배열의 관계

- 배열 이름의 정체
  - 배열 이름은 Compile시의 Symbol로서 첫 번째 요소의 주소 값을 나타낸다.
    - Symbol로서 컴파일시에만 유효함
    - 실행시에는 메모리에 잡히지 않음 (변수의 이름과 동일함)
  - cf.) Pointer는 실행시 메모리에 할당됨
    - 32비트 운영체제에서는 4바이트 할당

13-1 포인터와 배열의 관계

```
int main(void)
{
    int arr[3]={0, 1, 2};
    printf("배열의 이름: %p \n", arr);
    printf("첫 번째 요소: %p \n", &arr[0]);
    printf("두 번째 요소: %p \n", &arr[1]);
    printf("세 번째 요소: %p \n", &arr[2]);
    // arr = &arr[i];  // 이 문장은 컴파일 에러를 일으킨다.
    return 0;
}
```

실행결과

```
배열의 이름: 0012FF50
첫 번째 요소: 0012FF50
두 번째 요소: 0012FF54
세 번째 요소: 0012FF58
```

배열 요소간 주소 값의 크기는 4바이트임을 알 수 있다 (모든 요소가 붙어있다는 의미).

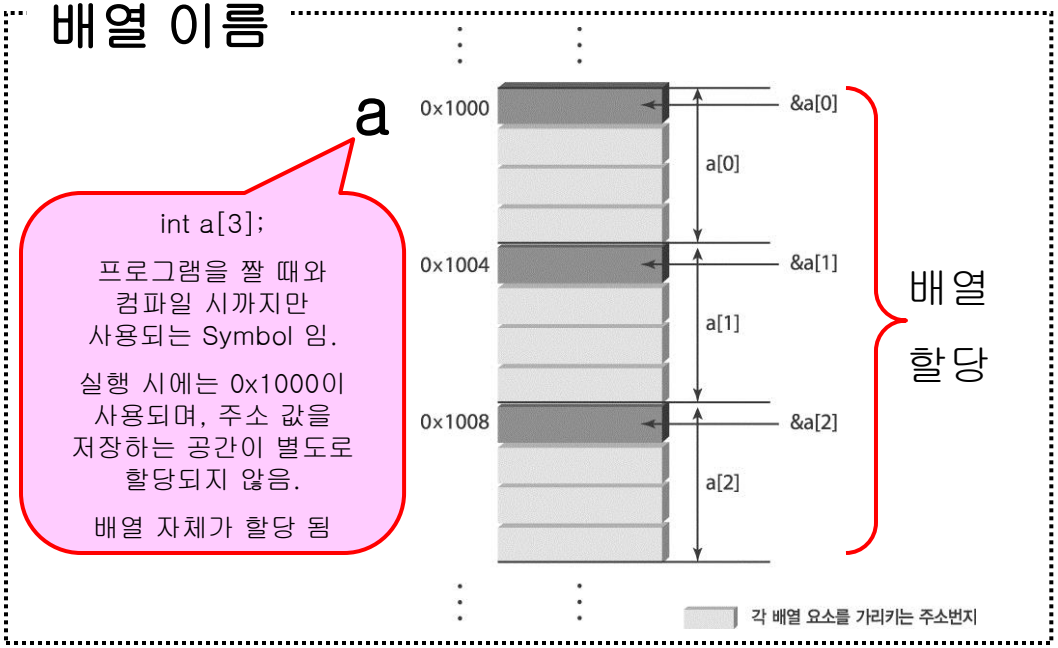
Programming in C, 정재은 저, 사이텍미디어. 9장 참조

(교재의 13-1은 읽지 말 것)

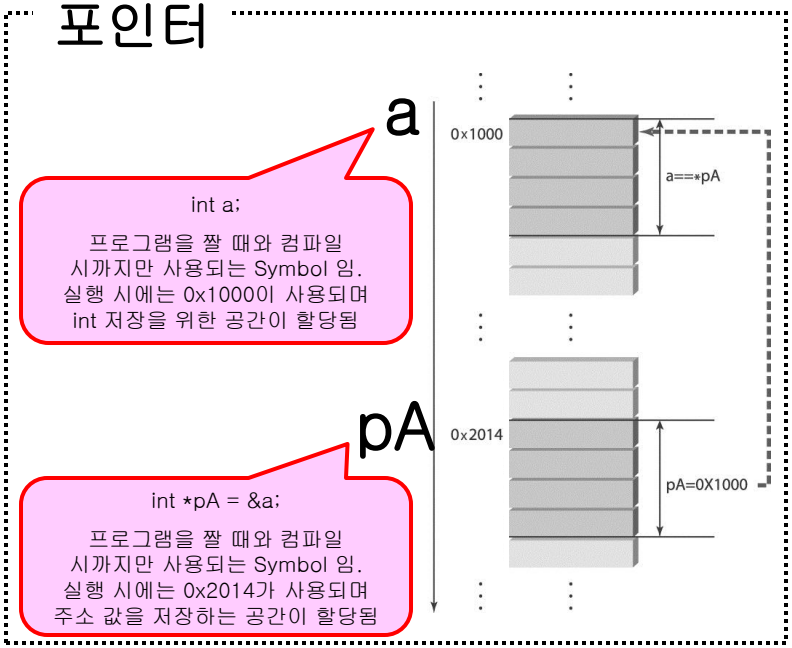
배열 이름과 포인터 비교

- 포인터처럼 실행 시에 배열 이름에 해당하는 메모리(주소 값 저장)가 할당되는가?
  - No: 포인터와 달리 실행 시 메모리에 주소 값을 저장하기 위한 공간이 할당되지 않음
    - 배열 자체가 할당됨

배열 이름



포인터



13-1 포인터와 배열의 관계

- 배열 이름

```
int main(void)
{
    int a[5]={0, 1, 2, 3, 4};
    int b=10;
    a=&b;    //Error!
}
```

a는 실행 시에 메모리에 존재하지 않음

실습

- 배열 이름을 포인터처럼 사용 가능

```
int main(void)
{
    int arr1[3]={1, 2, 3};
    double arr2[3]={1.1, 2.2, 3.3};

    printf("%d %g \n", *arr1, *arr2);
    *arr1 += 100;
    *arr2 += 120.5;
    printf("%d %g \n", arr1[0], arr2[0]);
    return 0;
}
```

실행결과

```
1 1.1
101 121.6
```

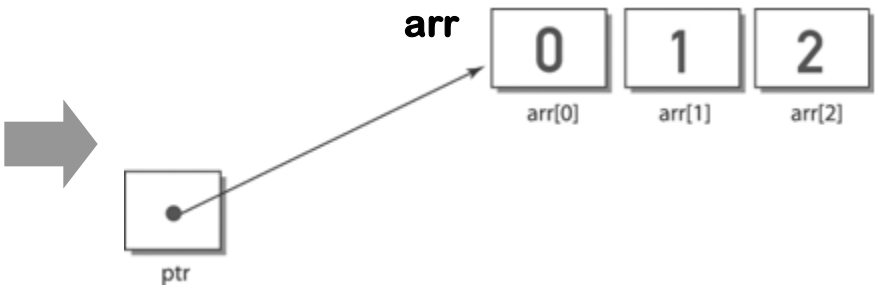
실습

- 포인터를 배열처럼 활용 가능
  - 포인터에 index 활용 가능

```
#include <stdio.h>

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

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



13-2 포인터 연산

- 포인터 연산이란?
  - 포인터가 저장한 값을 증가 혹은 감소시키는 연산을 의미

```
ptr1++;

ptr1 += 3;

--ptr1;

ptr2=ptr1+2;
```

그럴수도 있고 아닐 수도 있음

ptr의 타입에 따라 다름

- ptr=1000일 때, (ptr+1)은 1001일까?
- * 또는 / 연산은 허용될까?

안됨

13-2 포인터 연산

포인터 연산

- 포인터가 가리키는 대상의 자료형에 따라서 증가 및 감소되는 값이 차이를 지님

```
int main(void)
{
    int * ptr1=0x0010;
    double * ptr2=0x0010;

    printf("%p %p \n", ptr1+1, ptr1+2);
    printf("%p %p \n", ptr2+1, ptr2+2);

    printf("%p %p \n", ptr1, ptr2);
    ptr1++;
    ptr2++;
    printf("%p %p \n", ptr1, ptr2);
    return 0;
}
```

적절치 않은 초기화

왼편과 같이 포인터 변수에 저장된 값을 대상으로 하는 증가 및 감소연산을 진행할 수 있다.

실행결과

```
00000014 00000018
00000018 00000020
00000010 00000010
00000014 00000018
```

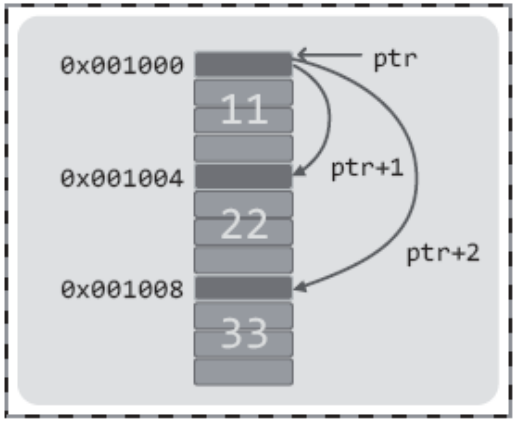
예제의 실행결과를 통해서 다음 사실을 알 수 있다.

- **int형 포인터 변수** 대상의 증가 감소 연산 시 **sizeof(int)**의 크기만큼 값이 증가 및 감소한다.
- **double형 포인터 변수** 대상의 증가 감소 연산 시 **sizeof(double)**의 크기만큼 값이 증가 및 감소한다.
- **type형 포인터 변수** 대상의 증가 감소 연산 시 **sizeof(type)**의 크기만큼 값이 증가 및 감소한다.

포인터 연산을 통한 배열 요소의 접근

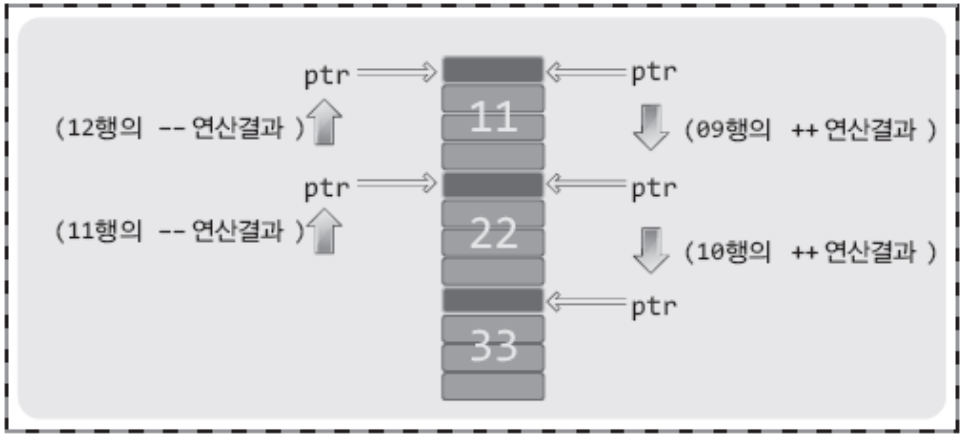
```
int main(void)
{
    int arr[3]={11, 22, 33};
    int * ptr=arr;    // int * ptr=&arr[0]; 과 같은 문장
    printf("%d %d %d \n", *ptr, *(ptr+1), *(ptr+2));

    printf("%d ", *ptr); ptr++;    // printf 함수호출 후, ptr++ 실행
    printf("%d ", *ptr); ptr++;
    printf("%d ", *ptr); ptr--;    // printf 함수호출 후, ptr-- 실행
    printf("%d ", *ptr); ptr--;
    printf("%d ", *ptr); printf("\n");
    return 0;
}
```




```
11 22 33
11 22 33 22 11
```

실행결과



int형 포인터 변수의 값은 4씩 증가 및 감소를 하니,

int형 포인터 변수가 int형 배열을 가리키면,

int형 포인터 변수의 값을 증가 및 감소시켜서

배열 요소에 순차적으로 접근이 가능하다.

포인터와 배열을 통해서 얻을 수 있는 결론

```
int main(void)
{
    int arr[3]={11, 22, 33};
    int * ptr=arr;
    printf("%d %d %d \n", *ptr, *(ptr+1), *(ptr+2));
    . . . .
}
```

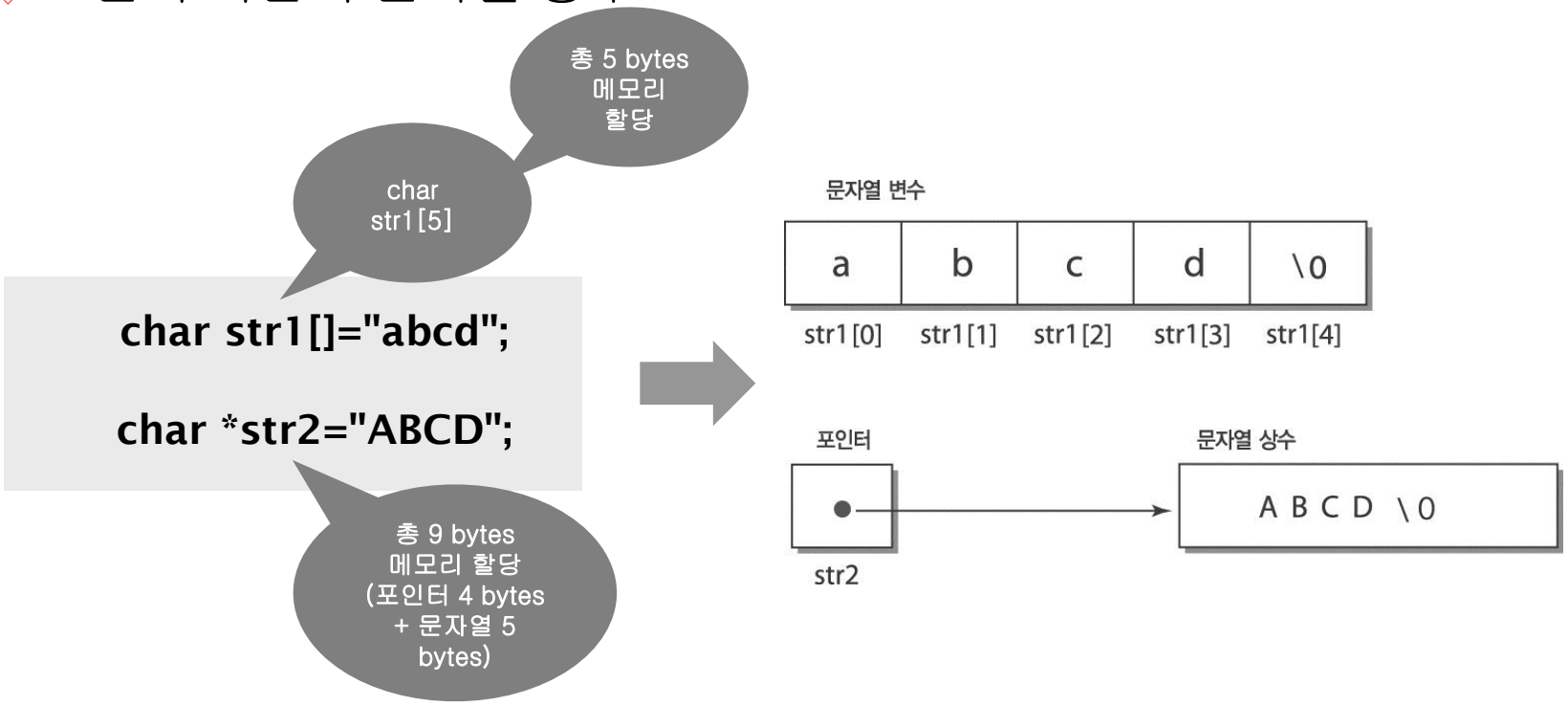
arr[i] == *(arr+i)

➡ arr이 "포인터"이거나 "배열 이름"인 경우

```
printf("%d %d %d \n", *(ptr+0), *(ptr+1), *(ptr+2));    // *(ptr+0)는 *ptr과 같다.
printf("%d %d %d \n", ptr[0], ptr[1], ptr[2]);
printf("%d %d %d \n", *(arr+0), *(arr+1), *(arr+2));    // *(arr+0)는 *arr과 같다.
printf("%d %d %d \n", arr[0], arr[1], arr[2]);
```

13-3 문자열 상수 포인터

- 문자열 표현 방식의 이해
  - 배열 기반의 문자열 변수
  - 포인터 기반의 문자열 상수



13-3 문자열 상수 포인터

```
#include <stdio.h>

int main()
{
    char str1[5]="abcd";          // 문자열 변수 선언
    char *str2="ABCD";            // 문자열상수 선언

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

    str1[0]='x';                  //문자열 변수 변경, OK
    str2[0]='x';                  //문자열 상수 변경, Error 발생

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

    return 0;
}
```

13-3 문자열 상수 포인터

- Try this!

```
#include <stdio.h>
int main(void)
{
    char *str1 = "Good!";
    char *str2 = "Good!";
    char str3[10] = “Good!”;

    printf("%d, %d, %d \n", str1, str2, str3);
    printf(“%s, %s, %s \n”, str1, str2, str3);
    return 0;
}
```

이건 상수! Data Segment (Section) 에 잡힘

이건 변수! Stack에 잡힘

이걸 char str2[] = “Good!”; 로 바꾸면?

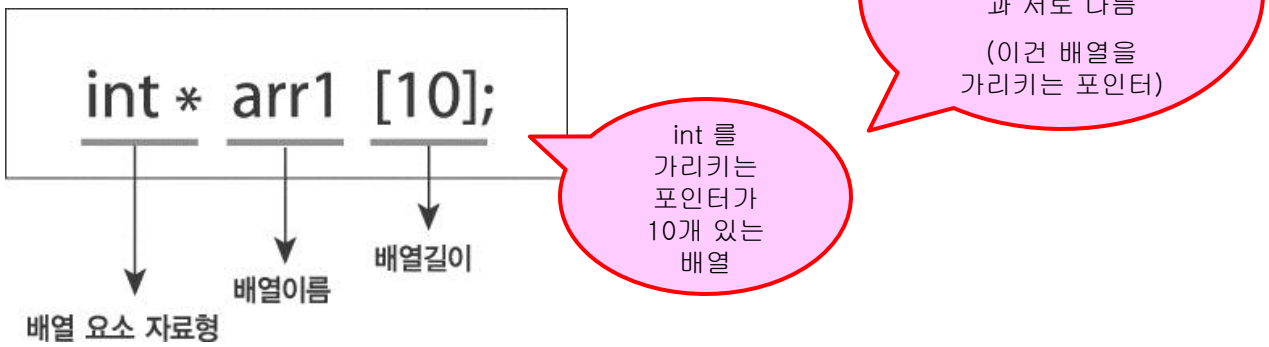
그럼 변수가 됨! Stack에 잡힘

- The result depends on the compiler.

13-4 포인터 배열

- 포인터 배열
  - 배열의 요소로 포인터를 지니는 배열

```
int* arr1[10];
double* arr2[20];
char* arr3[30];
```



  - 연산자 우선순위를 체크하자!

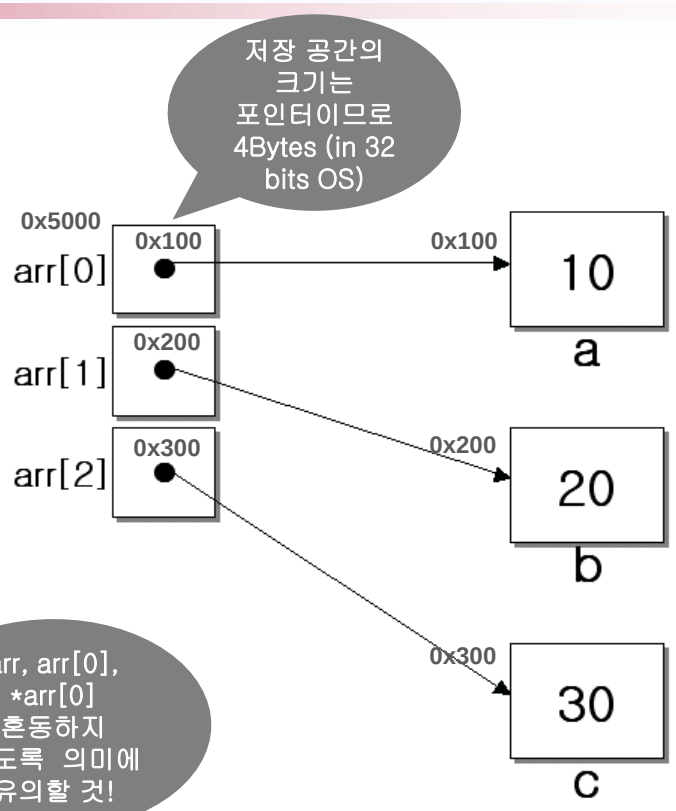
13-4 포인터 배열

- 포인터 배열 예제 1

```
#include <stdio.h>
 int main(void)
{
   int a=10, b=20, c=30;
   int* arr[3]={&a, &b, &c};

   printf("%d \n", *arr[0]);
   printf("%d \n", *arr[1]);
   printf("%d \n", *arr[2]);

   return 0;
}
```



arr, arr[0], *arr[0] 혼동하지 않도록 의미에 유의할 것!

표현식	내용	값
arr	int를 저장하는 arr 배열의 시작 주소	0x5000
arr[0]	int를 저장하는 arr 배열의 첫 번째 공간(4Bytes)에 저장되어 있는 변수(a)의 시작 주소 (&a)	0x100
*arr[0]	int를 저장하는 arr 배열의 첫 번째 공간(4Bytes)에 저장되어 있는 변수(a)의 시작 주소에 저장된 값 (a)	10

13-4 포인터 배열

- 포인터 배열 예제 2: 문자열 배열

```
#include <stdio.h>
 int main(void)
{
   char* arr[3]={
         "Fervent-lecture",
         "TCP/IP",
         "Socket Programming"
   };
   printf("%s \n", arr[0]);
   printf("%s \n", arr[1]);
   printf("%s \n", arr[2]);
   return 0;
}
```

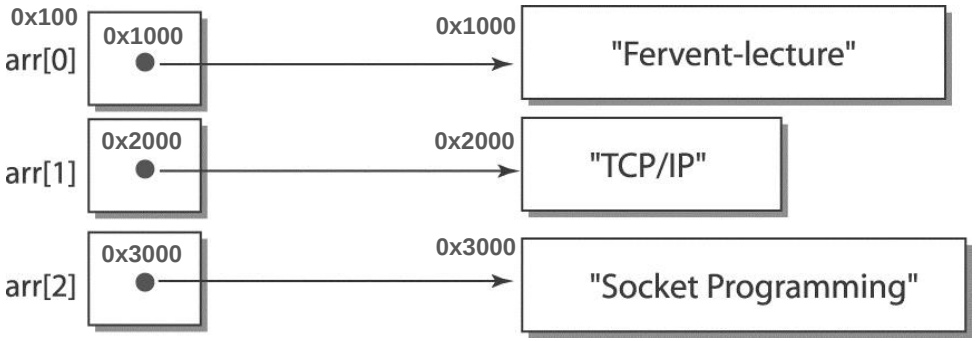
원래, (문자열이 아닌) 문자가 저장된 주소를 가리키는 포인터의 배열

문자열의 시작 문자의 주소를 가리키고 있음을 알고 있으므로 %s를 사용함 (원래 타입대로 문자라면 %c를 사용했어야 함)

char * arr[3] = {"Fervent-lectur","TCP/IP","Socket Programming"};

↓

char * arr[3] = {0x1000, 0x2000, 0x3000};



원래, char *는 문자가 저장된 주소를 가리키지만,

보통, char *는 문자열의 시작 주소를 가리키는 의미로 주로 사용됨

연습문제

- 포인터를 이용해 배열을 조작해 보자.
  - 문자열을 입력 받아 그 길이를 출력하는 프로그램을 작성하자. 이 때, 널 문자는 길이에 포함하지 않으며, 길이를 계산하는 과정에서 포인터를 사용해야 한다.
  - 문자열을 입력 받아 거꾸로 뒤집은 다음 결과를 출력하는 프로그램을 작성하자. 이 때, 뒤집는 과정은 포인터를 사용해야 한다.