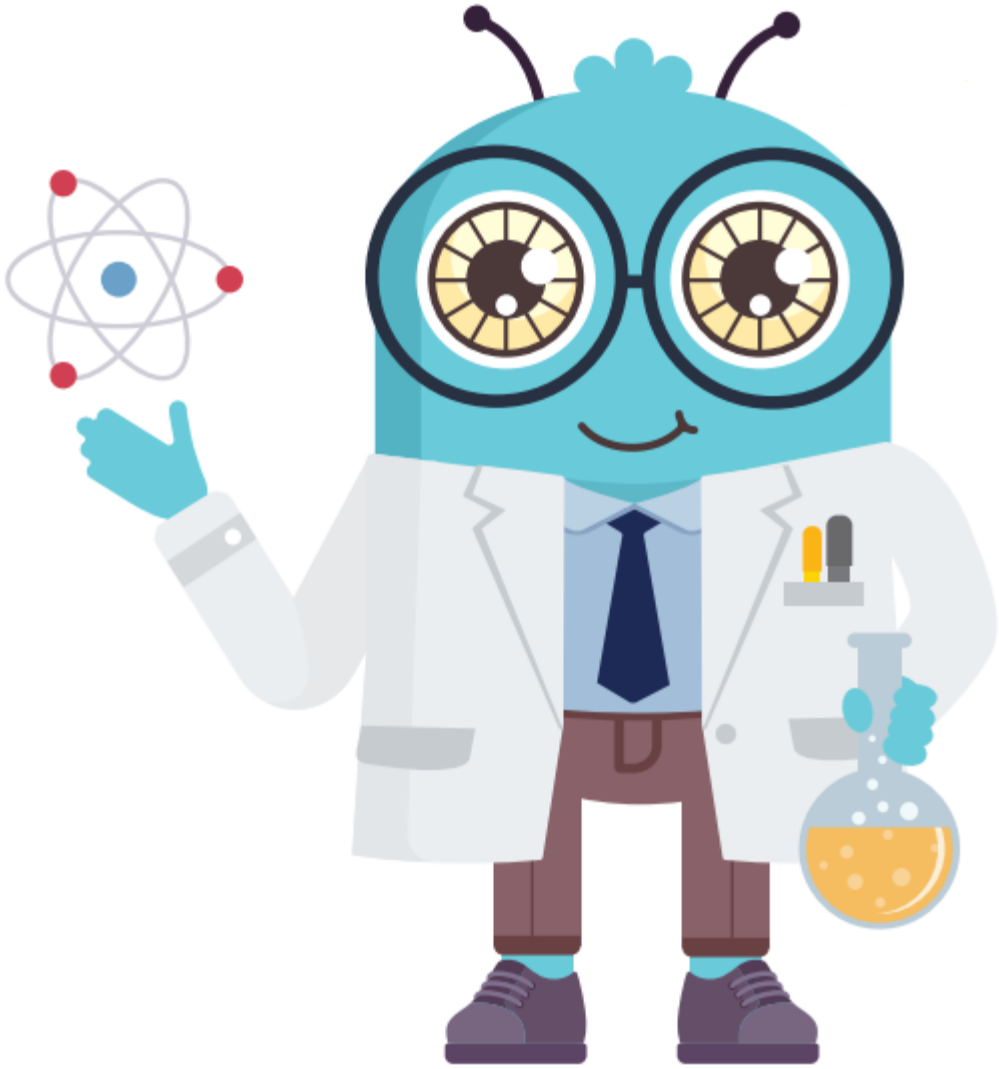


Chapter 09

배열과 포인터

C 언어 for Beginner(4판)

목차

01

스택

1. 스택의 이해

- 한쪽 끝이 막혀있는 구조
- 가장 먼저 들어간 것이 가장 나중에 나옴 : LIFO(Last In First Out)

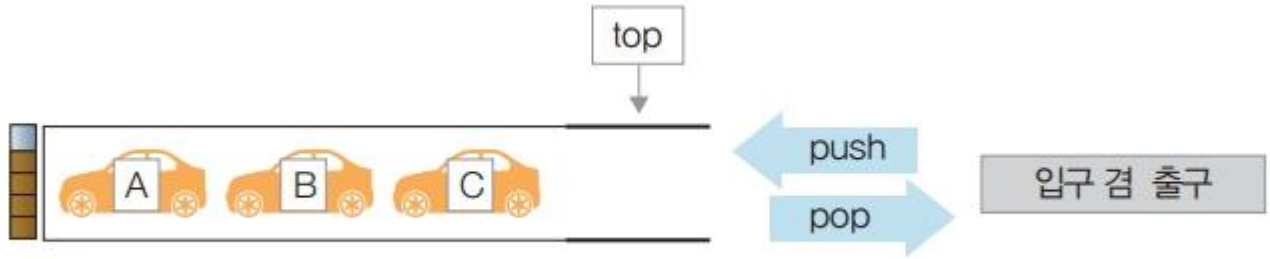


그림 9-1 스택의 기본 개념

- 용어
  - top(탑) : 가장 마지막에 들어간 데이터의 위치를 가리킴
  - push(푸쉬) : 데이터를 넣는 것
  - pop(팝) : 데이터를 빼는 것

2. 배열로 스택 만들기

- 자동차 5대가 들어가지만 한쪽이 막힌 터널 만들기
- 초기화

```
char stack[5];
int top=0;
```

  - [그림 9-2]와 같이 다섯 자리 배열이 잡히면 이 배열을 막힌 터널(스택)이라고 가정
  - 현재 자동차가 없으므로 top이 0을 가리키고 있음

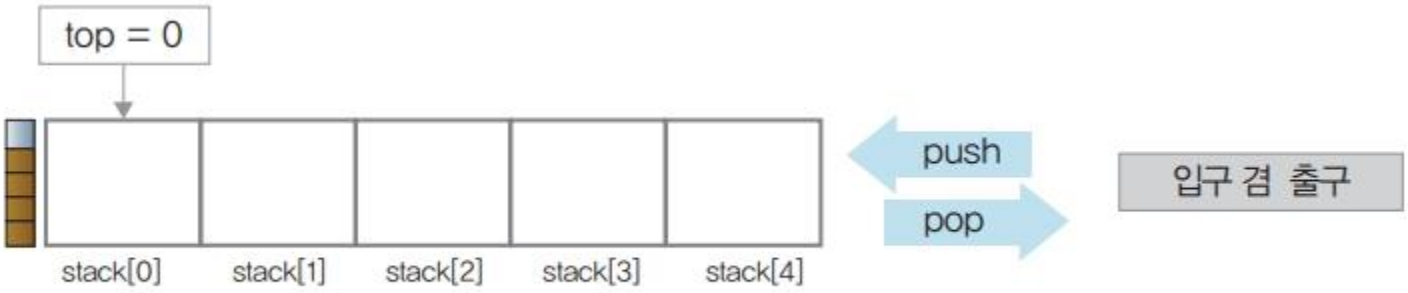


그림 9-2 비어 있는 터널

- **'자동차 A'를 넣기(push)**
  - 자동차 A를 넣으면 top은 0에서 1로 바뀌고 위치는 stack[0]에서 stack[1]로 이동

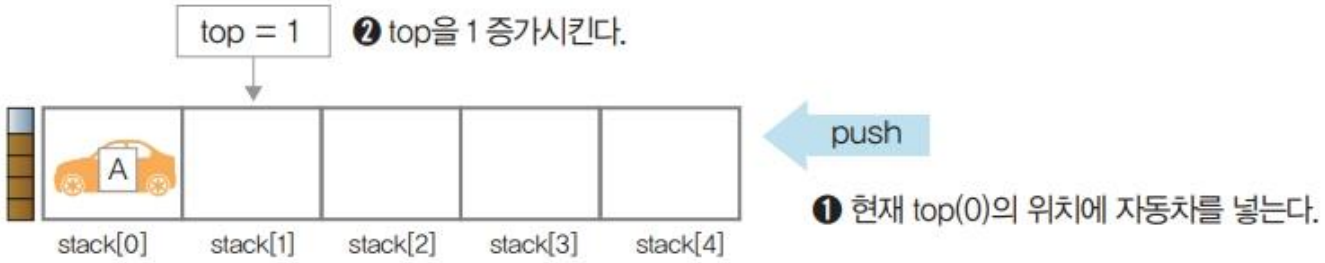


그림 9-3 터널에 자동차 1대 넣기

- **'자동차 B'와 '자동차 C'를 터널에 넣기**
  - top은 3이 되어 stack[3]의 위치로 이동

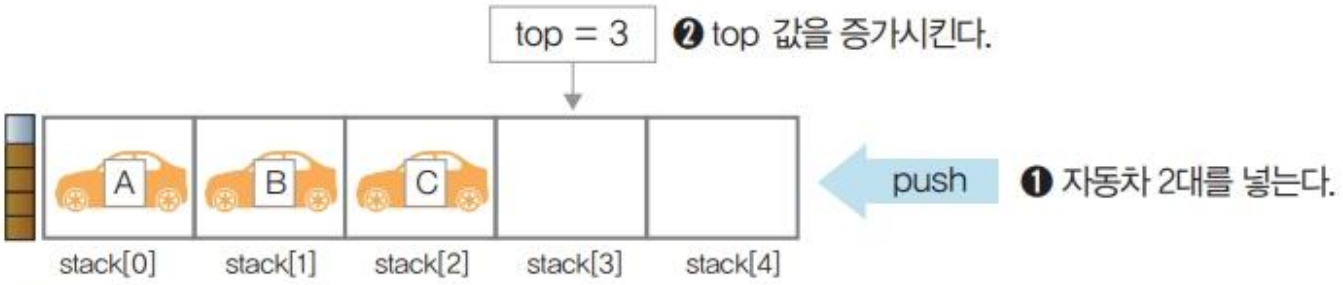


그림 9-4 터널에 자동차 3대 넣기

- **자동차 1대 빼기(pop)**
  - 자동차 1대를 뺄 때는 top을 1 감소시킨 후 그 자리의 자동차를 빼내면 됨

그림 9-5 터널에서 자동차 1대 빼기

1. 스택의 이해

기본 9-1 스택 구현 예 1 — 9-1.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    char stack[5];
06    int top=0;
07
08    stack[top] = 'A';
09    printf(" %c 자동차가 터널에 들어감\n", stack[top]);
10    top ++;
11
12    stack[top] = 'B';
13    printf(" %c 자동차가 터널에 들어감\n", stack[top]);
14    top ++;
15
16    stack[top] = 'C';
17    printf(" %c 자동차가 터널에 들어감\n", stack[top]);
18    top ++;
```

05~06: 스택과 top의 초깃값을 선언한다.

08~10: 스택에 값이 들어간 후 top 값이 1만큼 증가한다.

12~14: 스택에 값이 들어간 후 top 값이 1만큼 증가한다.

16~18: 스택에 값이 들어간 후 top 값이 1만큼 증가한다.

1. 스택의 이해

```
19
20    printf("\n");
21
22    top --;                                          ----- top 값을 1씩 줄이면서 스택에서
23    printf(" %c 자동차가 터널을 빠져나감\n", stack[top]);      값을 하나씩 빼낸다.
24    stack[top] = ' ';
25
26    top --;                                          ----- top 값을 1씩 줄이면서 스택에서
27    printf(" %c 자동차가 터널을 빠져나감\n", stack[top]);      값을 하나씩 빼낸다.
28    stack[top] = ' ';
29
30    top --;                                          ----- top 값을 1씩 줄이면서 스택에서
31    printf(" %c 자동차가 터널을 빠져나감\n", stack[top]);      값을 하나씩 빼낸다.
32    stack[top] = ' ';
33 }
```

실행 결과

```
A 자동차가 터널에 들어감
B 자동차가 터널에 들어감
C 자동차가 터널에 들어감

C 자동차가 터널을 빠져나감
B 자동차가 터널을 빠져나감
A 자동차가 터널을 빠져나감
```

1. 스택의 이해

- 만약 top이 0일 때 자동차를 빼라는 명령을 받으면 오류 발생
- top이 5일 때 자동차를 넣으라고 하면 오류 발생
- 이렇게 오류까지 처리하려면 자동차가 들어가는 8~10행을 다음과 같이 수정
- 12~14행, 16~18행도 수정

```
if(top >= 5)
{
    printf("터널이 꽉 차서 차가 못 들어감.\n");
}
else
{
    stack[top] = 'A';
    printf(" %c 자동차가 터널에 들어감\n", stack[top]);
    top ++;
}
```

1. 스택의 이해

- 자동차가 빠져나가는 22~24행은 다음과 같이 수정
- 마찬가지로 26~28행, 30~32행도 수정

```
if(top <= 0)
{
   printf("현재 터널에 자동차가 없음\n");
}
else
{
   top --;
   printf(" %c 자동차가 터널을 빠져나감\n", stack[top]);
   stack[top] = ' ';
}
```

1. 스택의 이해

응용 9-2 스택 구현 예 2 — 9-2.c

```
01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03 void main( )
04 {
05    char stack[5];
06    int top=0;
07
08    char carName = 'A';          ---- 자동차 이름을 A부터 시작한다.
09    int select=9;                ---- 사용자가 선택할 작업을 입력할 변수이다.
10
11    while(select != 3)           ---- 사용자가 3을 선택하지 않으면 while문을 반복한다.
12    {
13       printf("<1> 자동차 넣기 <2> 자동차 빼기 <3> 끝 : ");
14       scanf("%d", &select);     ---- 사용자가 선택하는 값이다.
15
16       __1__
17       {
```

1. 스택의 이해

```
18      case 1:                                          ---- 사용자가 1(넣기)을
19        if(__2__)                                           선택하면 실행된다.
20        {  printf("터널이 꽉 차서 차가 못 들어감\n");  }   ------ 터널에 자동차 5대가
21        else                                                있으면 못 들어간다.
22        {                                                ------ 빈 곳이 있을 경우
23          stack[top] = carName++;                              (5대 미만이면)
24          printf(" %c 자동차가 터널에 들어감\n", stack[top]);   자동차를 넣고 top 값을
25          top ++;                                              1 증가시킨다.
26        }
27        break;                                           ------ switch문을 벗어난다.
28
29      case 2:                                            ---- 사용자가 2(빼기)를
30        if(__3__)                                           선택하면 실행된다.
31        {  printf("빠져나갈 자동차가 없음\n");  }          ------ 터널에 자동차가 1대도
32        else                                                없으면 빼낼 것이 없다.
33        {                                                ------ 빼낼 자동차가 있으면
34          top --;                                              (1대 이상이면) top 값을
35          printf(" %c 자동차가 터널에서 빠짐\n", stack[top]);   1 감소시키고 자동차를
36          stack[top] = ' ';                                    빼낸다. 그리고 그
37        }                                                      자리를 빈칸으로 채운다.
38        break;
```

1. 스택의 이해

```
39
40      case 3:                                          ---- 사용자가 3(끝)을
41          printf("현재 터널에 %d대가 있음.\n", top);          선택하면 현재 자동차
42          printf("프로그램을 종료합니다.\n");                 수를 출력하고 종료한다.
43          break;
44
45      default:                                         ---- 사용자가 1, 2, 3 외의
46          printf("잘못 입력했습니다. 다시 입력하세요. \n");     값을 입력하면 처리된다.
47      }
48  }
49  }
```

실행 결과

```
<1> 자동차 넣기 <2> 자동차 빼기 <3> 끝 : 1
 A 자동차가 터널에 들어감
<1> 자동차 넣기 <2> 자동차 빼기 <3> 끝 : 1
 B 자동차가 터널에 들어감
<1> 자동차 넣기 <2> 자동차 빼기 <3> 끝 : 2
 B 자동차가 터널에서 빠짐
<1> 자동차 넣기 <2> 자동차 빼기 <3> 끝 : 2
 A 자동차가 터널에서 빠짐
<1> 자동차 넣기 <2> 자동차 빼기 <3> 끝 : 2
빠져나갈 자동차가 없음
<1> 자동차 넣기 <2> 자동차 빼기 <3> 끝 : 3
현재 터널에 0대가 있음.
프로그램을 종료합니다.
```

02
메모리와 주소

1. 정수형 변수의 메모리 할당

```
int a = 100;
int b = 200;
```

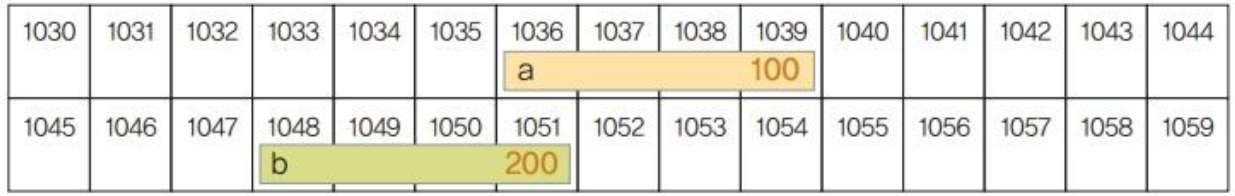


그림 9-6 메모리에 할당된 정수형 변수의 위치 예

- 메모리는 바이트(Byte) 단위로 나뉘며, 각 바이트에는 주소가 지정됨.
- 정수형 변수의 크기는 4바이트이므로 이 메모리에 정수형 변수 a를 선언하면 임의의 위치에 4바이트가 자리잡음.
- 변수가 위치하는 곳 : 주소(address)
- 변수의 주소를 알려면 변수 앞에 '&'를 붙임
  - a의 주소(&a) = 1036번지, b의 주소(&b) = 1040번지

1. 정수형 변수의 메모리 할당

기본 9-3 변수의 주소 알아내기 9-3.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    int a = 100;
06    int b = 200;
07
08    printf("변수 a의 주소는 %d 입니다. \n", &a);
09    printf("변수 b의 주소는 %d 입니다. \n", &b);
10 }
```

----- a와 b의 주소를 출력한다.

실행 결과

```
변수 a의 주소는 20184760 입니다.
변수 b의 주소는 20184748 입니다.
```

2. 정수형 배열의 메모리 할당

```
int aa[3] = { 10, 20, 30 };
```

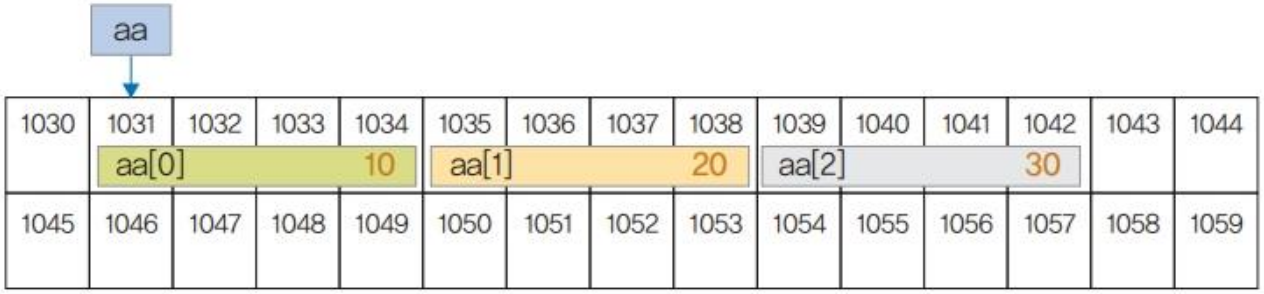


그림 9-7 메모리에 할당된 정수형 배열의 위치 예

- 배열의 주소 표현
  - aa[0]의 주소(&aa[0]) = 1031번지
  - aa[1]의 주소(&aa[1]) = 1035번지
  - aa[2]의 주소(&aa[2]) = 1039번지
- 배열 이름 aa = 전체 배열의 주소 = 1031번지

  배열 aa의 주소를 구할 때는 '&'를 쓰지 않고, 단순히 'aa'로 표현

2. 정수형 배열의 메모리 할당

기본 9-4 정수형 배열의 메모리 할당 1 9-4.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    int aa[3] = {10, 20, 30};
06
07    printf("aa[0]의 값은 %d, 주소는 %d \n", aa[0], &aa[0]);
08    printf("aa[1]의 값은 %d, 주소는 %d \n", aa[1], &aa[1]);
09    printf("aa[2]의 값은 %d, 주소는 %d \n", aa[2], &aa[2]);
10    printf("배열 이름 aa의 값(=주소)는 %d \n", aa);
11 }
```

----- 배열의 각 자리 값과 주소를 출력한다.

----- 배열 이름(aa 주소)을 출력한다.

실행 결과

```
aa[0]의 값은 10, 주소는 13629916
aa[1]의 값은 20, 주소는 13629920
aa[2]의 값은 30, 주소는 13629924
배열이름 aa의 값(=주소)는 13629916
```

2. 정수형 배열의 메모리 할당

- aa+1에서 +1은 단순히 1을 더하라는 의미가 아니라 '배열 aa의 위치에 서 한 칸 건너뛰라'는 의미
- '한 칸'은 현재 aa가 정수형 배열이므로 4바이트를 의미

표 9-1 배열의 주소 표현

배열 첨자로 표현	배열 이름으로 표현	실제 주소
&aa[0]	aa + 0	1031
&aa[1]	aa + 1	1035
&aa[2]	aa + 2	1039

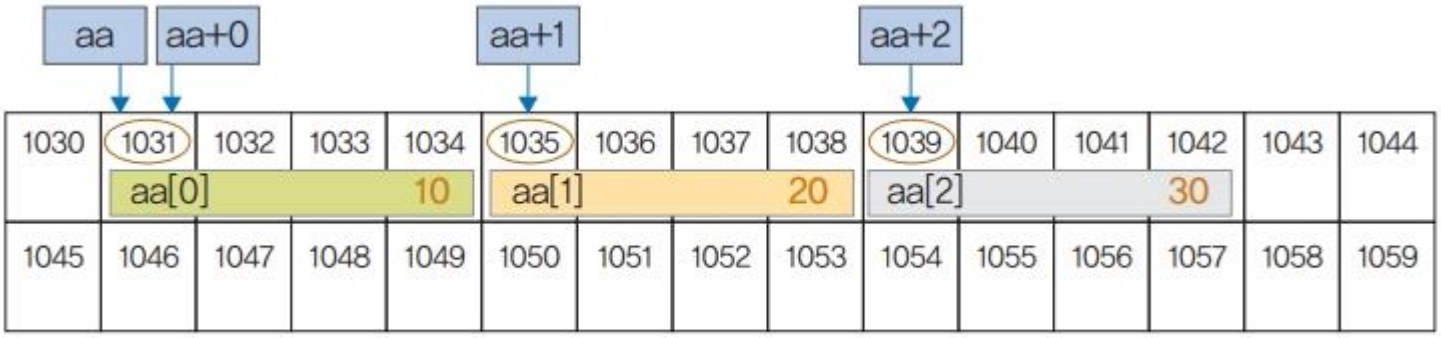


그림 9-8 배열 이름으로 주소 표현

2. 정수형 배열의 메모리 할당

응용 9-5 정수형 배열의 메모리 할당 2 — 9-5.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    int aa[3] = {10, 20, 30};
06
07    printf("&aa[0]는 %d, aa+0은 %d \n", __1__,   aa+0);      ----- &aa[0]==aa+0
08    printf("&aa[1]는 %d, aa+1은 %d \n", &aa[1], __2__);      ----- &aa[1]==aa+1
09    printf("&aa[2]는 %d, aa+2는 %d \n", &aa[2], __3__);      ----- &aa[2]==aa+2
10 }
```

정답_ 1 &aa[0] 2 aa+1 3 aa+2

실행 결과

```
&aa[0]는 7601340, aa+0은 7601340
&aa[1]는 7601344, aa+1은 7601344
&aa[2]는 7601348, aa+2는 7601348
```

03
포인터

2. 정수형 배열의 메모리 할당

- 포인터란 주소를 담는 그릇(변수)

그림 9-9 변수의 종류

- 포인터 선언 : * 를 붙여줌

❶ char ch;	실행 결과 ▶	문자형 변수를 선언한다.
❷ char* p;	실행 결과 ▶	문자형 포인터 변수를 선언한다.
❸ ch = 'A'	실행 결과 ▶	문자형 변수에 문자 'A'를 대입한다.
❹ p = &ch;	실행 결과 ▶	포인터 변수에 변수 ch의 주소인 '&ch'를 대입한다.

2. 정수형 배열의 메모리 할당

- 이 구문을 실행하면 다음과 같이 메모리에 변수가 자리 잡음

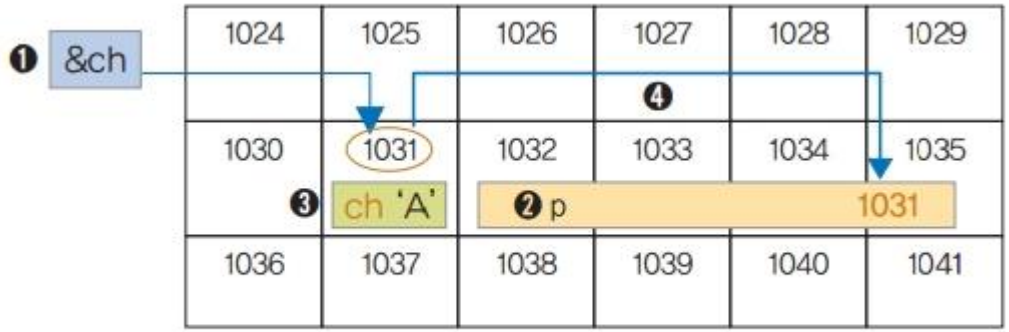


그림 9-10 일반 변수와 포인터 변수의 관계 예

- ❶의 문자형 변수 ch는 1바이트를 차지하므로 주소 1031번지에 1바이트가 자리 잡음(&ch 는 1031을 뜻하는 주솟값
- ❷의 포인터 변수(char*) p는 1032~1035번지에 4바이트가 자리 잡음(포인터 변수는 크기가 4바이트)
- ❸의 변수 ch에 'A 값'을 넣고 ❹의 포인터 변수 p에 변수 ch의 주솟값인 &ch를 넣음
- &ch는 1031번지를 의미하므로 포인터 변 수 p에는 1031이 들어감

기본 9-6 일반 변수와 포인터 변수의 관계 9-6.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    char ch;                  ----- 문자형 변수와 포인터 변수를 선언한다.
06    char* p;
07
08    ch = 'A';                 ----- 문자 'A'를 ch에 대입하고 ch의 주소를 p에 대입한다.
09    p = &ch;
10
11    printf("ch가 가지고 있는 값: ch  ==> %c \n", ch);
12    printf("ch의 주소(address): &ch ==> %d \n", &ch);
13    printf("p가 가지고 있는 값 : p   ==> %d \n", p);
14    printf("p가 가리키는 곳의 실제 값 : *p  ==> %c \n", *p);
15 }
```

실행 결과

```
ch가 가지고 있는 값: ch  ==> A
ch의 주소(address): &ch ==> 9042587
p가 가지고 있는 값 : p   ==> 9042587
p가 가리키는 곳의 실제 값 : *p  ==> A
```

3. 포인터

- 13행에서는 변수 ch의 주솟값을 넣었으므로 12행과 동일한 9042587이 출력
- 핵심은 14행인데 *p는 'p에 저장된 주소(9042587)가 가리키는 곳의 실제 값'이라고 이해
- 9042587번지에 들어 있는 'A'가 출력

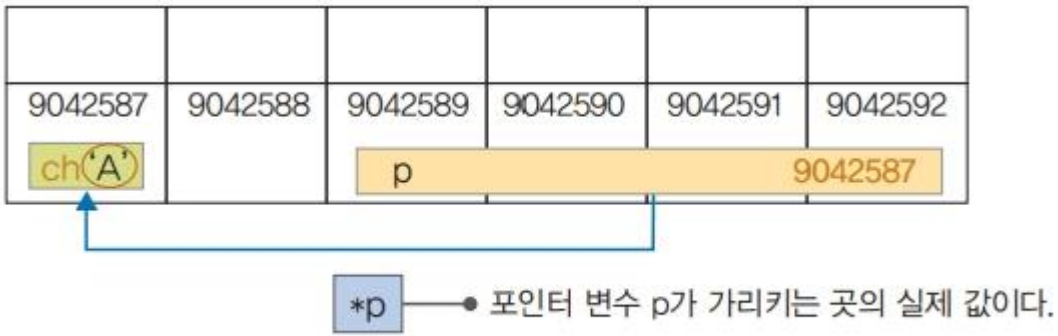


그림 9-11 [기본 9-6]의 변수와 포인터의 관계

- ❶ 포인터 변수를 선언하려면 변수형에 * 기호를 붙여야 함
- 즉 int*, char*, float*와 같이 쓰면 포인터 변수
- int*는 정수형 포인터 변수, char*는 문자형 포인터 변수, float* 는 실수형 포인터 변수

- ❷ char* p;로 선언하면 p에 문자형 변수의 주솟값을 넣어야 하고 int* p;로 선언하면 p에 정수형 변수의 주솟값을 넣어야 함

`int a;`	실행 결과 ▶	정수형 변수 a를 선언한다.
`char* p;`	실행 결과 ▶	문자형 포인터 변수 p를 선언한다.
`p = &a;`	실행 결과 ▶	문자형 포인터 변수에 정수형 변수의 주소를 넣는다. (×)

`int a;`	실행 결과 ▶	정수형 변수 a를 선언한다.
`int* p;`	실행 결과 ▶	정수형 포인터 변수 p를 선언한다.
`p = &a;`	실행 결과 ▶	정수형 포인터 변수에 정수형 변수의 주소를 넣는다. (O)

여기서 잠깐 포인터 변수의 크기

- 포인터 변수는 정수형이든 문자형이든 무조건 4바이트를 차지

응용 9-7 포인터의 관계 이해하기 9-7.c

```c
#include <stdio.h>

void main( )
{
    char ch;
    char* p;
    char* q;

    ch = 'A';
    p = __❶__;

    q = p;

    __❷__ = 'Z';

    printf("ch가 가지고 있는 값: ch  ==> %c \n\n", ch);
}
```

- char ch; — 문자형 변수 ch를 선언한다.
- char* p; char* q; — 문자형 포인터 변수 p와 q를 선언한다.
- ch = 'A'; — ch에 문자를 대입한다.
- p = ❶; — ch의 주솟값을 p에 대입한다.
- q = p; — p의 값을 q에 대입한다.
- ❷ = 'Z'; — q가 가리키는 곳의 실제 값을 변경한다.

실행 결과

```
ch가 가지고 있는 값: ch  ==> Z
```

정답 ❶ &ch ❷ *q

3. 포인터

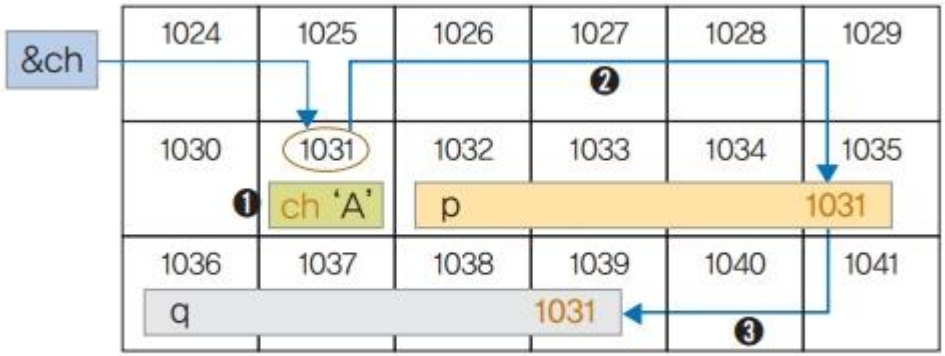


그림 9-12 [응용 9-7]의 변수와 포인터의 관계 1

- 5~7행에서 문자형 변수 ch, 문자형 포인터 변수 p와 q를 선언하면 각각 1031번지, 1032~ 1035번지, 1036~1039번지의 자리를 차지
- 9행에서는 ch에 'A'를 대입하고(❶) 10행에서 는 ch의 주솟값(&ch, 1031번지)을 포인터 변수 p에 대입(❷)
- 12행에서 p의 값을 q에 대입(❸)

3. 포인터

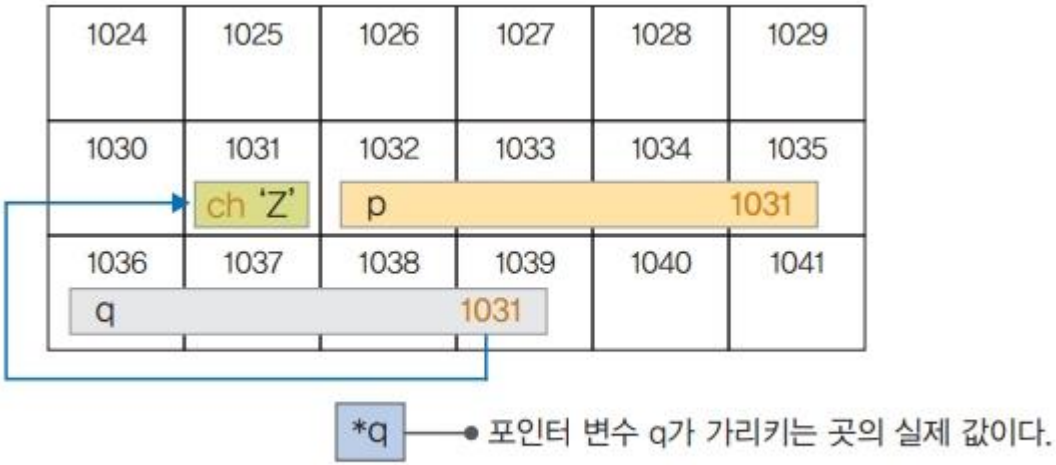


그림 9-13 [응용 9-7]의 변수와 포인터의 관계 2

- 14행은 'q가 가리키는 곳의 실제 값을 Z로 변경하라'는 의미이므로(*q는 q가 가리키는 곳의 실제 값) [그림 9-13]과 같이 변경
- 결국 q가 가리키는 곳은 1031번지의 실제 값이므로 'A' 가 'Z'로 변경된 것
- 16행에서 ch를 출력하면 'Z'가 출력

여기서 잠깐 포인터 변수를 정의하는 * 기호의 위치

- 포인터를 정의할 때 * 기호는 데이터형에 붙이든 변수에 붙이든 관계없음

04
배열과 포인터의 관계

4. 배열과 포인터의 관계

1. 문자형 배열과 포인터

- 배열을 s[12]라고 선언하면 s는 변수가 아닌 주솟값이 됨

기본 9-8 문자형 배열과 포인터의 관계 1 — 9-8.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    char s[8]= "Basic-C";        ------ 문자형 배열을 선언하고 초깃값을 대입한다.
06    char  *p;                    ------ 문자형 포인터 변수를 선언한다.
07
08    p = s;                       ------ p에 배열 s의 주소를 대입한다.
09
10    printf("&s[3] ==> %s\n", &s[3]);   ------ 문자열과 포인터의 주솟값을 %s로 출력한다.
11    printf("p+3   ==> %s\n\n", p+3);
12
13    printf("s[3]   ==> %c\n", s[3]);   ------ 문자와 포인터의 실제 값을 %c로 출력한다.
14    printf("*(p+3) ==> %c\n", *(p+3));
15 }
```

실행 결과

```
&s[3] ==> ic-C
p+3   ==> ic-C

s[3]   ==> i
*(p+3) ==> i
```

1. 문자형 배열과 포인터

- [기본 9-8]의 5행에서 여덟 자리 문자형 배열 s를 선언하고 "Basic-C"라는 문자열로 초기화
- 6행에서 문자형 포인터 변수 p를 선언하고 8행에서 p에 배열 s의 주솟값인 s를 대입
- [기본 9-8]의 변수와 포인터의 관계를 그림으로 나타내면 다음과 같음

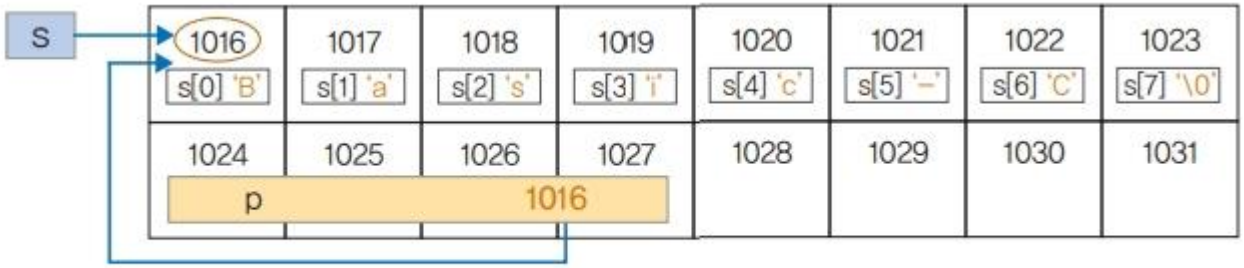


그림 9-14 [기본 9-8]의 변수와 포인터의 관계

1. 문자형 배열과 포인터

- 현재 포인터 변수 p에는 1016이 들어 있으므로 11행의 p+3은 거기서 세 칸을 건너뛴 1019
- 1019번지에는 i가 들어 있으므로 10행과 마찬가지로 'ic-C'가 출력
- 13행에서는 s[3]을 printf("%c")로 출력하니 당연히 'i'가 출력
- 14행의 *(p+3) 역시 p에서 세 칸을 건너뛴 주소의 실제 값을 의미하므로 1019번지의 실제 값인 'i'가 출력

표 9-2 배열의 포인터 표현

```
char s[3];
char* p;
p = s;
```

배열 첨자	포인터 상수	포인터 변수
s[0]	*(s+0)	*(p+0)
s[1]	*(s+1)	*(p+1)
s[2]	*(s+2)	*(p+2)

2. 문자열 배열과 포인터의 응용

응용 9-9 문자형 배열과 포인터의 관계 2 9-9.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    char s[8]= "Basic-C";
06    char  *p;
07    int i;
08
09    p = __1__ ;                  ----- 포인터 변수에 배열 주소를 대입한다.
10
11    for( i=sizeof(s)-2; i >= 0; i-- )   ----- 문자형 배열의 끝부터 배열의 개수만큼 반복한다.
12       printf("%c", *( __2__ ));        ------ 포인터 변수가 가리키는 곳의 문자 하나를 출력한다.
13
14    printf("\n");
15 }
```

정답_ 1 s 2 p+i

실행 결과

```
C-cisaB
```

2. 문자열 배열과 포인터의 응용

- 5행 : 문자열 배열 선언, 'Basic-C'로 초기화
- 6행 : 포인터 변수 p 선언
- 9행 : 배열 s의 이름을 p에 대입
- 11행 : i의 초기값을 '배열크기-2'로 대입

  i가 0보다 크거나 같은 동안 반복(6~0까지 7회 반복)
- 12행 : (p+i)의 실제값 출력 → *(p+i)

(p+6) → (p+5) → (p+4) → (p+3) → (p+2) → (p+1) → (p+0)

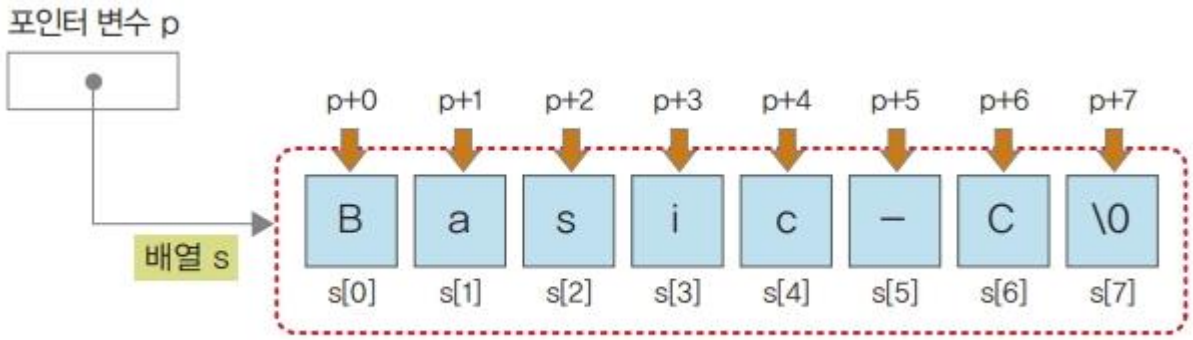


그림 9-15 [응용 9-9]의 변수와 포인터의 관계

3. 포인터 학습 노하우

❶ 포인터 변수가 무엇을 가리키는지 확인

❷ 포인터 변수를 선언할 때는 변수 앞에 *만 붙임

```
int *a;
char *str;
float *b;
```

❸ 포인터 변수에는 꼭 주솟값을 넣어야 함

- 변수 이름 앞에 '&' 를 붙임
- 배열의 이름은 그 자체가 주소이므로 '&'를 붙이지 않음

❶ 변수인 경우

```
int a;
int *p;

p = &a;
```

❷ 배열인 경우

```
int a[10];
int *p;

p = a;
```

4. 배열과 포인터의 관계

❹ **포인터가 가리키는 곳의 실제값을 구하려면 *를 붙임**

- 포인터 변수 p가 변수 a가 들어있는 주소인 1016번지를 가리킨다고 가정

```
int a=123;
int *p;
p = &a;
```

`printf("%d", p)`	실행결과 ▶	a의 주소인 1016이 출력된다(주소 자체는 중요하지 않다).
`printf("%d", *p)`	실행결과 ▶	p가 가리키는 곳의 실제값, 즉 a값인 123이 출력된다.

그림 9-16 포인터가 가리키는 실제 값

4. 배열과 포인터의 관계

- *p에는 임의의 값을 대입할 수 있지만, p에는 오직 주소만 들어간다는 점에 주의

```
int a=123;
int *p;
```

p=&a;	실행결과 ▶	p에 a의 주솟값을 대입한다.(O)
*p = 1	실행결과 ▶	p가 가리키는 주소의 실제값을 정수 1로 바꾼다.(O)
p = 1	실행결과 ▶	p에 1번지라는 주솟값을 직접 넣으라는 의미다. 그런데 과연 컴퓨터의 메모리에 1번지라는 주소가 있을까? 또 있다고 해도 전혀 엉뚱한 위치가 된다.(×)

*

예제 모음

[예제모음 24] 포인터를 이용해 문자열을 거꾸로 출력

예제 설명 8장의 [예제모음 20]에서처럼 입력한 문자열을 반대 순서로 출력해보자. 이번에는 포인터를 활용하여 작성한 프로그램이다.

실행 결과

```
문자열을 입력하세요 : CookBook
내용을 거꾸로 출력 ==> kooBkooC
```

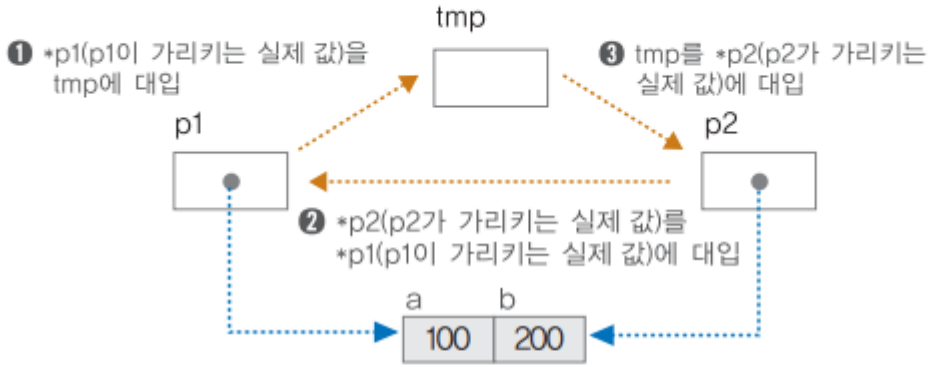
[예제모음 24] 포인터를 이용해 문자열을 거꾸로 출력

IT COOKBOOK

```
01 #define  _CRT_SECURE_NO_WARNINGS
02 #include <string.h>
03 #include <stdio.h>
04 void main( )
05 {
06    char ss[100];                  ----- 입력받을 문자 배열을 선언한다.
07    int count, i;
08    char *p;                       ----- 문자형 포인터를 선언한다.
09
10    printf("문자열을 입력하세요 : ");
11    scanf("%s", ss);               ----- 문자열을 입력한다.
12
13    count = strlen(ss);            ----- 입력한 문자열의 개수이다.
14
15    p = ss;                        ----- 배열 ss의 주소를 포인터 변수 p에 대입한다.
16
17    printf("내용을 거꾸로 출력 ==> ");
18    for(i=0; i < count; i++)       ----- 포인터 p에 있는 실제 값을 문자열의 맨
19    {                                    뒤부터 출력한다.
20       printf("%c", *(p+count-(i+1)));
21    }
22    printf("\n");
23 }
```

[예제모음 25] 포인터를 이용한 두 값의 교환

예제 설명 입력한 두 값을 포인터를 활용하여 교환하는 프로그램이다. 값을 바꾸는 개념은 다음과 같다.



실행 결과

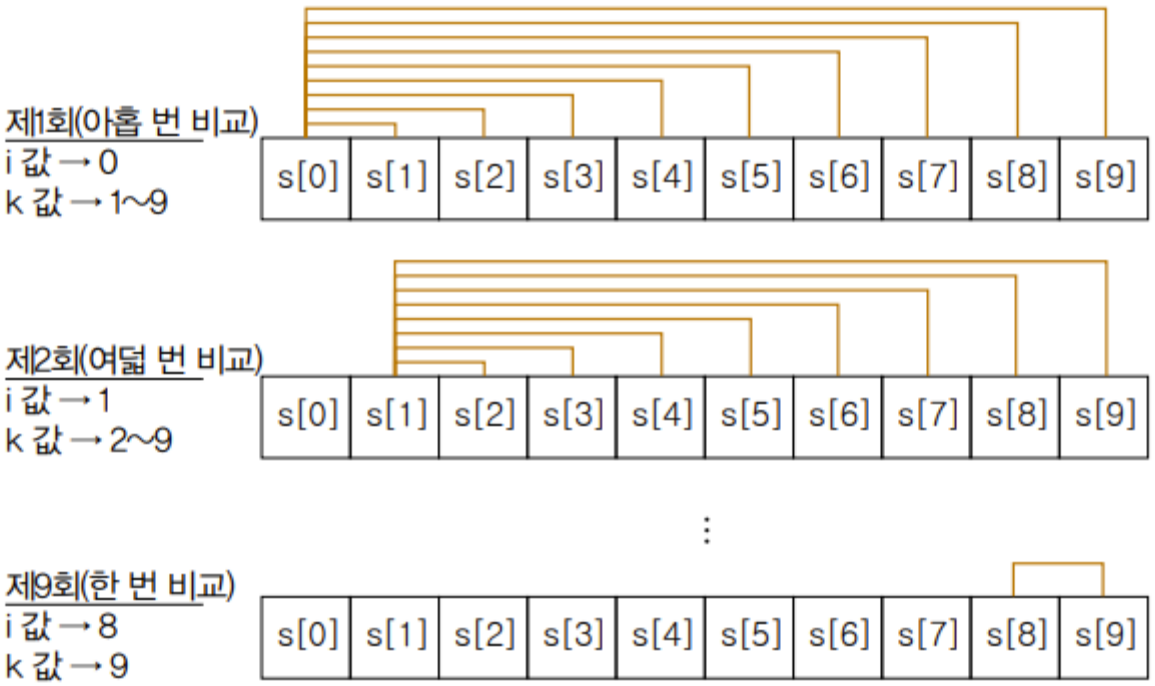
```
a 값 입력 : 100
b 값 입력 : 200
바뀐 값 a는 200, b는 100
```

[예제모음 25] 포인터를 이용한 두 값의 교환


```
01 #define  _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03 void main( )
04 {
05    int a, b, tmp;          ----- 정수형 변수 3개와 포인터 변수 2개를 선언한다.
06    int *p1, *p2;
07
08    printf("a 값 입력 : ");   ----- a와 b에 값을 입력한다.
09    scanf("%d", &a);
10    printf("b 값 입력 : ");
11    scanf("%d", &b);
12
13    p1 = &a;                ----- 변수 a의 주솟값을 p1에 대입한다.
14    p2 = &b;                ----- 변수 b의 주솟값을 p2에 대입한다.
15
16    tmp = *p1;              ----- ❶ p1이 가리키는 곳의 실제 값을 tmp에 넣는다.
17    *p1 = *p2;              ----- ❷ p2가 가리키는 곳의 실제 값을 p1이 가리키는 곳에 넣는다.
18    *p2 = tmp;              ----- ❸ tmp에 저장된 값을 p2가 가리키는 곳에 넣는다.
19
20    printf("바뀐 값 a는 %d, b는 %d \n", a, b);
21 }
```

[예제모음 26] 포인터를 이용한 배열의 정렬

예제 설명 포인터를 이용하여 배열에 들어 있는 값 10개를 정렬하는 프로그램이다. 다음 그림을 참고하여 두 값을 비교하고 작은 것을 앞쪽으로 옮기는 선택 정렬을 사용한다.



실행 결과

```
정렬 전 배열 s ==> 1 0 3 2 5 4 7 6 9 8
정렬 후 배열 s ==> 0 1 2 3 4 5 6 7 8 9
```

[예제모음 26] 포인터를 이용한 배열의 정렬


```
01 #include <stdio.h>
02
03 void main( )
04 {
05    int s[10] = {1, 0, 3, 2, 5, 4, 7, 6, 9, 8};
06    int tmp;
07    int i, k;
08
09    int *p;
10
11    p = s;
12
13    printf("정렬 전 배열 s ==> ");
14    for(i=0; i < 10; i++)
15    {
16       printf("%d  ", *(p+i));
17    }
18    printf("\n");
19
```

배열 s에 정숫값 10개를 초기화하고 관련 변수를 선언한다.

포인터 변수를 선언한다.

배열 s의 주소를 포인터 변수 p에 대입한다.

정렬 전의 상태로 데이터 10개를 출력한다.

[예제모음 26] 포인터를 이용한 배열의 정렬

IT COOKBOOK

```
20    for(i=0; i < 9; i++)
21    {
22       for(k=i+1; k < 10; k++)
23       {
24          if( *(p+i) > *(p+k) )
25          {
26             tmp = *(p+i);
27             *(p+i) = *(p+k);
28             *(p+k) = tmp;
29          }
30       }
31    }
32    printf("정렬 후 배열 s ==> ");
33    for(i=0; i < 10; i++)
34    {
35       printf("%d  ", *(p+i));
36    }
37    printf("\n");
38 }
```

9회 반복한다. 내부 for문으로 비교할 두 값 중 첫 번째 값의 자리는 s[0]~s[8]이다.

내부 for문으로 9회, 8회, 7회, …, 1회 반복한다. 비교할 두 값 중 두 번째 값의 자리는 s[1]~s[9]이다.

p+i의 실제 값이 p+k의 실제 값보다 크면 두 값을 바꾼다.

정렬 후의 상태로 데이터 10개를 출력한다.

감사합니다!