C프로그래밍

11 배열의 응용과 포인터

10 예제 review

[예제 02] 성적 총점 계산 프로그램

영희, 철수, 영철, 영수 등 4명의 학생의 국어, 영어, 수학, 과학 점수(1~9점)를 입력 받아 배열에 저장하고, 각 학생의 총점과 과목별 총점을 계산하여 저장하고, 출력하는 프로그램을 작성하시요.

<1단계> 점수를 입력 받아서 저장

	국어	영어	수학	과학	학생별 총점
영희	2	3	4	5	
철수	5	5	4	7	
영철	3	7	2	4	
영수	8	6	2	9	
과목별 총점					

<2단계> 총점을 계산/저장하고 출력

	국어	영어	수학	과학	학생별 총점
영희	2	3	4	5	14
철수	5	5	4	7	21
영철	3	7	2	4	16
영수	8	6	2	9	25
과목별 총점	18	21	12	25	

1. 입력 받은 점수와 총점은 5X5 2차원 배열에 저장하여 출력하시요.

[예제 02] 성적 총점 계산 프로그램

영희, 철수, 영철, 영수 등 4명의 학생의 국어, 영어, 수학, 과학 점수(1~9점)를 입력 받아 배열에 저장하고, 각 학생의 총점과 과목별 총점을 계산하여 저장하고, 출력하는 프로그램을 작성하시요.

(1) 영희, 철수, 영철, 영수 등 4명의 학생 이름 저장

(2) 국어, 영어, 수학, 과학 등 4개의 과목 이름 저장

(3) 점수와 총점을 저장하는 배열 생성

(4) 각 학생별로 각 과목의 점수를 입력 받아 배열에 저장

(5) 총점을 계산하여 저장하고, 출력

[예제 02] 성적 총점 계산 프로그램

(1) 영희, 철수, 영철, 영수 등 4명의 학생 이름 저장

- ✓ 4명의 이름을 하나로 저장해서 관리할 수 없을까?
- ✓ 점수를 입력 받아 저장하는 5X5 배열과 각 학생의 이름을 연관 지을 수는 없을까?

➔ 4명의 학생 이름을 배열에 저장 : 0번 영희, 1번 철수, 2번 영철, 3번 영수

- ✓ 영희 ➔ 문자열이다. 그렇다면 문자열의 배열을 만들면 4명의 학생 이름을 저장 가능

➔ 문자열은 1차원 문자배열이다. 그렇다면 2차원 문자배열을 만들면 가능

char name[ ][10] = {"영희", "철수", "영철", "영수"};

name[0] ➔ "영희", name[1] ➔ "철수"

[예제 02] 성적 총점 계산 프로그램

(2) 국어, 영어, 수학, 과학 등 4개의 과목 이름 저장

- ✓ 4개의 과목 이름을 배열에 저장 : 0번 국어, 1번 영어, 2번 수학, 3번 과학

char subject[ ][10] = {"국어", "영어", "수학", "과학"};

[예제 02] 성적 총점 계산 프로그램

(3) 점수와 총점을 저장하는 배열 생성

✓ 가로(행)은 학생의 점수를, 새로(열)은 과목의 점수를 나타내는 5X5 정수형 배열 생성

```
int score[5][5] = {0};
```

[예제 02] 성적 총점 계산 프로그램

(4) 각 학생별로 각 과목의 점수를 입력 받아 배열에 저장

✓ 학생별로 과목의 점수를 입력하도록 요청하여 배열에 저장

```
for(int i=0; i<4; i++){          // i는 가로(행)을 나타내며 학생에 대응
  for(int j=0; j<4; j++){        // j는 세로(열)을 나타내며 과목에 대응
    printf("%s 학생의 %s 과목 점수를 입력하세요 : ", name[i], subject[j]);
    scanf("%d", &score[i][j]);
    printf("\n");
} }
```

```
영희 학생의 국어 점수를 입력하세요 : 5
영희 학생의 영어 점수를 입력하세요 : 4
영희 학생의 수학 점수를 입력하세요 : 7
영희 학생의 과학 점수를 입력하세요 : 2
철수 학생의 국어 점수를 입력하세요 : 7
.....
```

[예제 02] 성적 총점 계산 프로그램

(5) 총점을 계산하여 저장하고, 출력

✓ 학생별로 총점을 출력하고, 과목별로 총점을 출력함

```
// 학생별로 총점을 계산하여 저장하고, 출력하는 예시
int total;
for(int i=0; i<4; i++){  // i는 가로(행)을 나타내며 학생에 대응
        total = 0;
        for(int j=0; j<4; j++){  // j는 세로(열)을 나타내며 과목에 대응
            total = total + score[i][j]; }
        score[i][j] = total; // 각 학생의 총점을 [0~3][4]에 저장
        printf("%s의 총점은 %d 입니다.\n", name[i], score[i][j]);
}
```

01
스택

스택의 이해

- **한쪽 끝이 막혀 있는 구조**
  - ✓ 가장 먼저 들어간 것이 가장 나중에 나옴 : LIFO(Last In First Out)

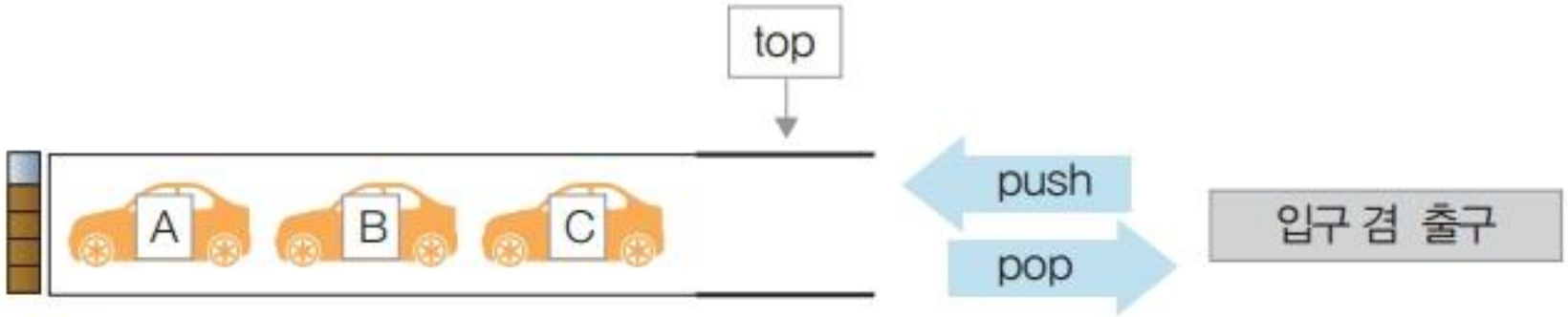


그림 9-1 스택의 기본 개념

- **용어**
  - ✓ top : 새로운 데이터가 들어갈 위치를 가리킴
  - ✓ push : 데이터를 넣는 것
  - ✓ pop : 데이터를 빼는 것

배열로 스택 만들기 (1/10)

- **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

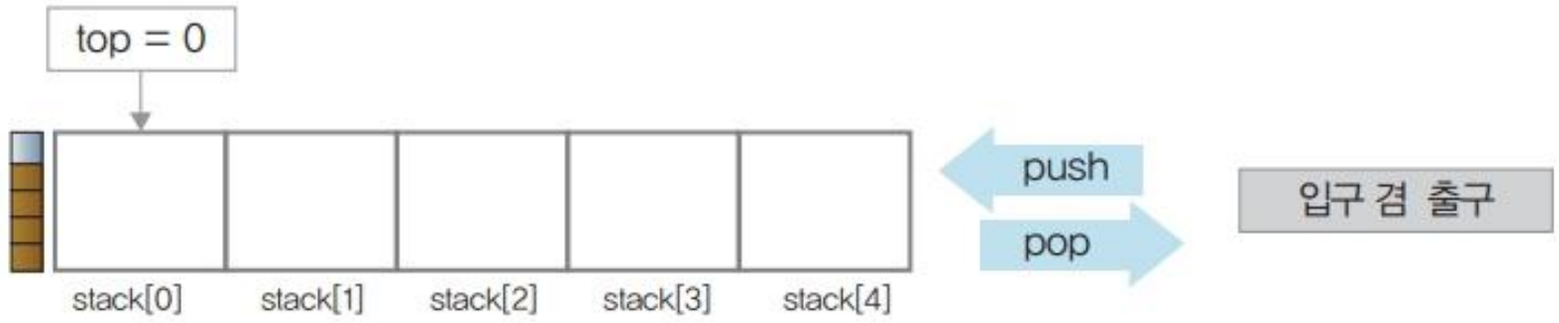


그림 9-2 비어 있는 터널

배열로 스택 만들기 (2/10)

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

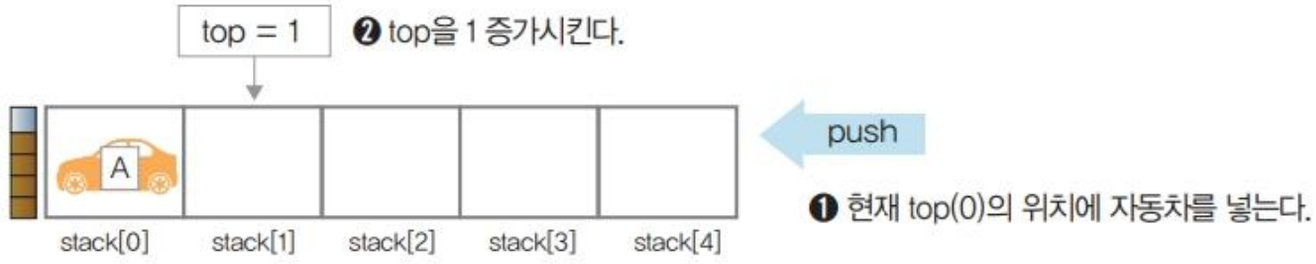


그림 9-3 터널에 자동차 1대 넣기

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

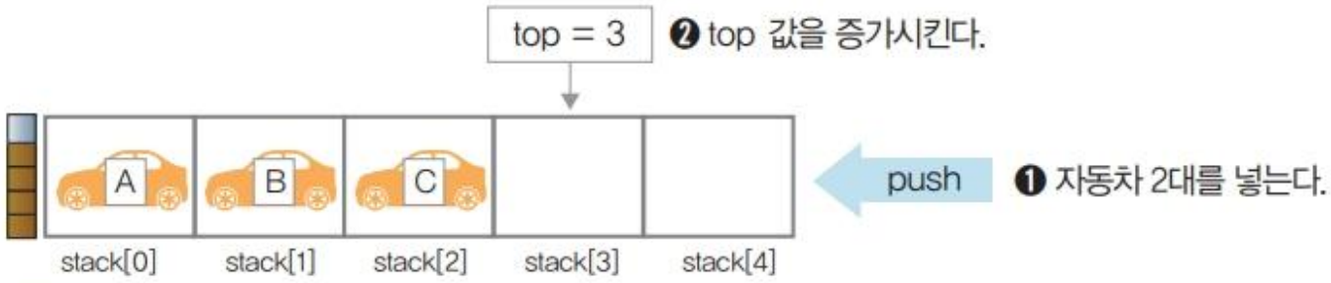


그림 9-4 터널에 자동차 3대 넣기

배열로 스택 만들기 (3/10)

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

그림 9-5 터널에서 자동차 1대 빼기

배열로 스택 만들기 (4/10)

- **자동차 3대(A, B, C)를 입고시키고, 다시 순서대로 3대를 출고 시키는 용량이 5인 자동차 보관 프로그램**

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.

배열로 스택 만들기 (5/10)

- **자동차 3대(A, B, C)를 입고시키고, 다시 순서대로 3대를 출고 시키는 용량이 5인 자동차 보관 프로그램 (cont'd)**

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

top 값을 1씩 줄이면서 스택에서 값을 하나씩 빼낸다.

top 값을 1씩 줄이면서 스택에서 값을 하나씩 빼낸다.

top 값을 1씩 줄이면서 스택에서 값을 하나씩 빼낸다.

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
```

배열로 스택 만들기 (6/10)

- **기본 9-1 보완**
  - ✓ 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 ++;
}
```

배열로 스택 만들기 (7/10)

- **기본 9-1 보완 (cont'd)**
  - ✓ 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] = ' ';
}
```

배열로 스택 만들기 (8/10)

- **스택 구조를 가지는 용량이 5인 주차장 프로그램**

  ✓ 차량은 A, B, C,… 순으로 입고

응용 9-2 스택 구현 예 2

```
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       {
```

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.
```

배열로 스택 만들기 (9/10)

- **스택 구조를 가지는 용량이 5인 주차장 프로그램 (cont'd)**

```
18      case 1:                                  ---- 사용자가 1(넣기)을
19        If(top>=5)                                  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(top<=0)                                  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;
```

배열로 스택 만들기 (10/10)

- **스택 구조를 가지는 용량이 5인 주차장 프로그램 (cont'd)**

```
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.
```

정답_1 switch(select) 2 top >= 5 3 top <= 0

02
메모리와 주소

정수형 변수의 메모리 할당 (1/2)

- **메모리는 바이트(Byte) 단위로 나뉘며, 각 바이트에는 주소가 지정**
  - ✓ 정수형 변수의 크기가 4바이트일 때
  - ➔ 정수형 변수 a를 선언하면 임의의 메모리 위치에 4바이트가 자리잡음
  - ✓ 변수가 위치하는 곳 : 변수의 값이 저장된 곳의 첫 번째 바이트 주소

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

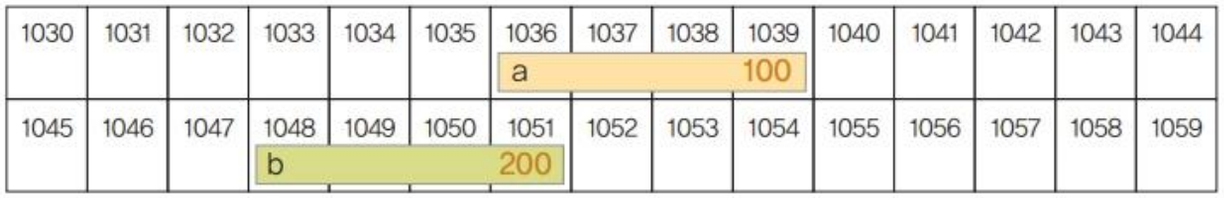


그림 9-6 메모리에 할당된 정수형 변수의 위치 예

- **변수의 주소를 알려면 변수 앞에 '&'를 붙임**
  - ✓ a의 주소(&a) = 1036번지, b의 주소(&b) = 1040번지

정수형 변수의 메모리 할당 (2/2)

- **변수의 주소를 출력하는 프로그램**

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.
```

정수형 배열의 메모리 할당 (1/4)

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

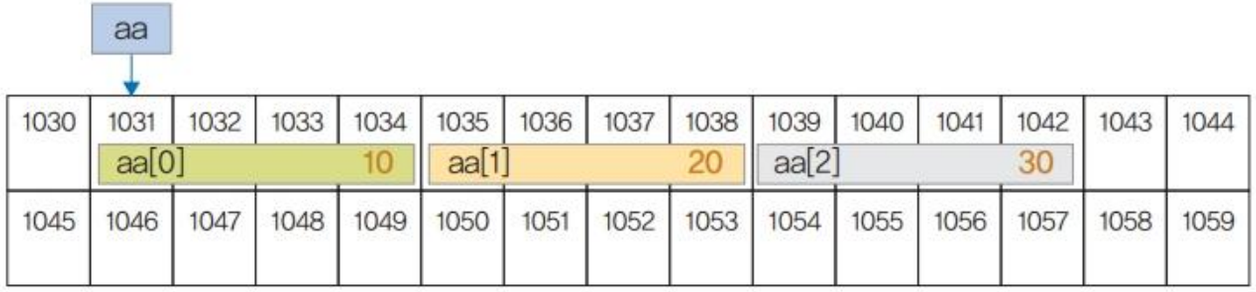


그림 9-7 메모리에 할당된 정수형 배열의 위치 예

- **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

  ➔ 배열의 이름은 배열의 전체 주소를 나타내는 역할

정수형 배열의 메모리 할당 (2/4)

- **배열의 주소를 출력하는 프로그램**

기본 9-4 정수형 배열의 메모리 할당 1

```
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
```

정수형 배열의 메모리 할당 (3/4)

- aa+1의 의미 : +1은 단순히 주소값에 1을 더하라는 의미가 아니라

  '배열 aa의 위치에 서 한 칸 건너뛰라'는 의미

- '한 칸'은 현재 aa가 정수형 배열이므로 4바이트를 의미

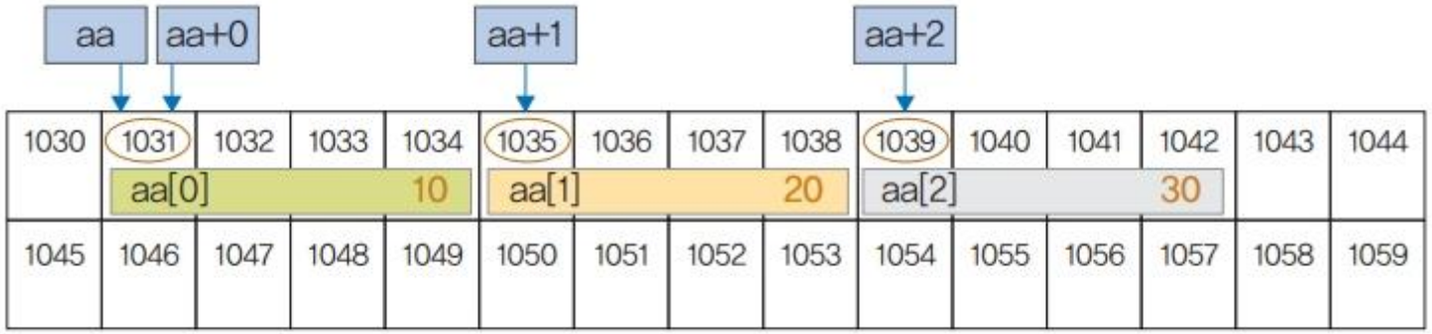


그림 9-8 배열 이름으로 주소 표현

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

정수형 배열의 메모리 할당 (4/4)

- **연산을 이용한 배열의 주소 출력 프로그램**

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", &aa[0], aa+0);      ----- &aa[0]==aa+0
08    printf("&aa[1]는 %d, aa+1은 %d \n", &aa[1], aa+1);      ----- &aa[1]==aa+1
09    printf("&aa[2]는 %d, aa+2는 %d \n", &aa[2], aa+2);      ----- &aa[2]==aa+2
10 }
```


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 개요

포인터 개요 (1/10)

- **포인터 : 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.

포인터 개요 (2/10)

- **전 페이지 코드의 실행 결과**

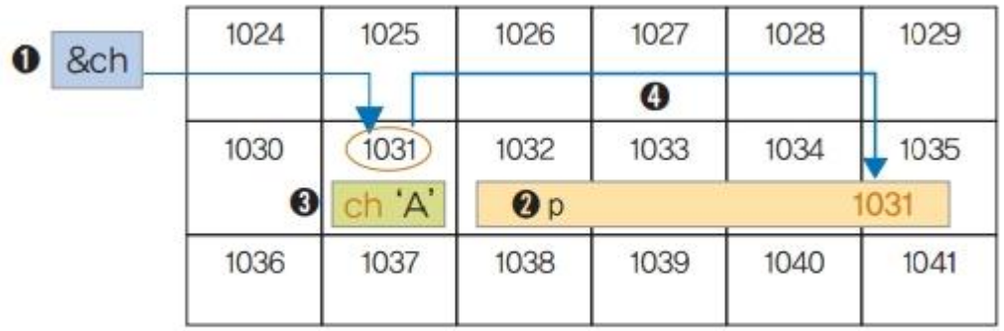


그림 9-10 일반 변수와 포인터 변수의 관계 예

❶의 문자형 변수 ch는 1바이트를 차지하므로 주소 1031번지에 1바이트가 자리 잡음

(&ch 는 1031을 뜻하는 주소의 값)

❷의 포인터 변수(char*) p는 1032~1035번지에 4바이트가 자리 잡음

(포인터 변수의 크기가 4바이트일 경우, 주소는 1031과 꼭 연속된 것은 아님)

❸의 변수 ch에 'A 값'을 넣고 ❹의 포인터 변수 p에 변수 ch의 주솟값인 &ch를 넣음

➔ &ch는 1031번지를 의미하므로 포인터 변수 p에는 1031이 들어감

포인터 개요 (3/10)

- **일반 변수와 포인터 변수의 출력 예시**

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
```

포인터 개요 (4/10)

- **기본 9-6 reviw**
  - ✓ 13행 : 변수 ch의 주솟값을 넣었으므로 12행과 동일한 90425870이 출력
  - ✓ 14행 : *p는 p에 저장된 주소(9042587)가 가리키는 곳의 실제 값’ 이라고 이해
  - ➔ 9042587번지에 들어 있는 ‘A’가 출력
  - ➔ 포인터 변수를 선언할 때도 ‘*’을 사용하고, 값을 가리킬 때도 ‘*’을 사용

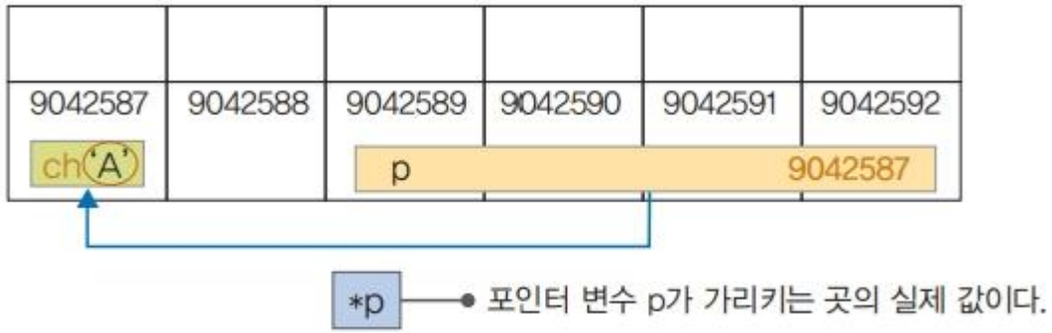


그림 9-11 [기본 9-6]의 변수와 포인터의 관계

&ch == p

ch == *p == ‘A’

포인터 개요 (5/10)

- **포인터 사용**

  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

포인터 개요 (6/10)

- **포인터 사용 (cont'd)**

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. (○)

여기서 잠깐 포인터 변수의 크기

- 포인터 변수는 정수형이든 문자형이든 동일한 크기를 가짐
- 포인터 변수의 크기가 무조건 4바이트인 것은 아니며, 운용 플랫폼과 컴파일러에 따라 달라지게 됨
- sizeof( ) 연산자를 이용해서 포인터 변수의 크기를 확인할 수 있음

포인터 개요 (7/10)

- **포인터 관계 이해 예시**

응용 9-7 포인터의 관계 이해하기 9-7.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    char ch;          ---- 문자형 변수 ch를 선언한다.
06    char* p;          ---- 문자형 포인터 변수 p와 q를 선언한다.
07    char* q;
08
09    ch = 'A';         ---- ch에 문자를 대입한다.
10    p = &ch;          ---- ch의 주솟값을 p에 대입한다.
11
12    q = p;            ---- p의 값을 q에 대입한다.
13
14    *q = 'Z';         ---- q가 가리키는 곳의 실제 값을 변경한다.
15
16    printf("ch가 가지고 있는 값: ch  ==> %c \n\n", ch);
17 }
```

실행 결과

```
ch가 가지고 있는 값: ch  ==> Z
```

포인터 개요 (8/10)

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

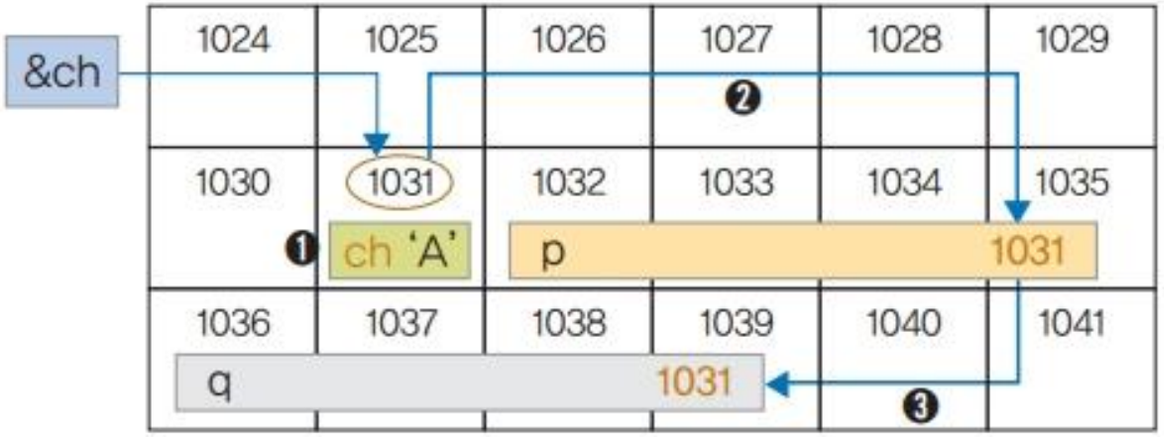


그림 9-12 [응용 9-7]의 변수와 포인터의 관계 1

포인터 개요 (9/10)

- **응용 9-7 review (cont'd)**
  - ✓ 14행은 'q가 가리키는 곳의 실제 값을 Z로 변경하라'는 의미
  - ➔ *q는 q가 가리키는 곳의 실제 값
  - ➔ q가 가리키는 곳은 1031번지의 실제 값이므로 'A' 가 'Z'로 변경된 것

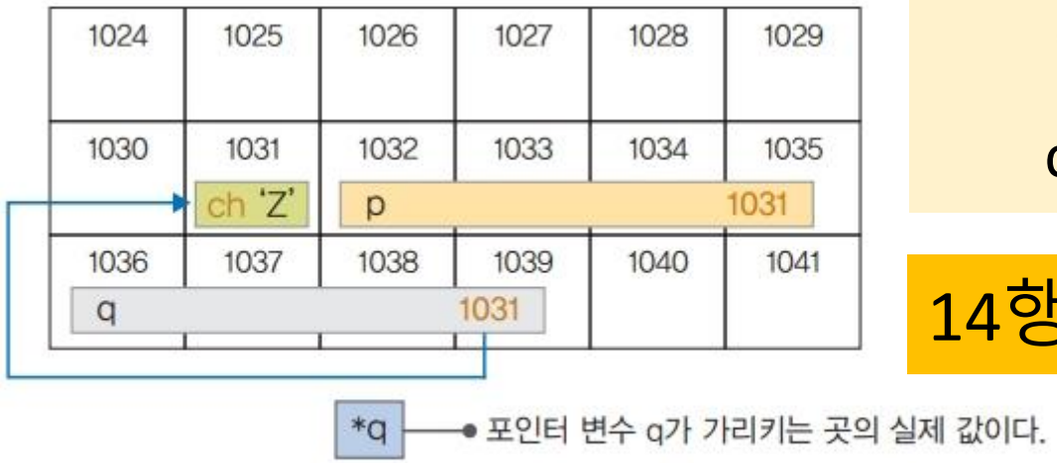


그림 9-13 [응용 9-7]의 변수와 포인터의 관계 2

&ch == p == q

ch == *p == *q

14행 : *q = 'Z'

포인터 개요 (10/10)

여기서 잠깐 포인터 변수를 정의하는 * 기호의 위치

- 포인터를 정의할 때 * 기호는 데이터형에 붙이든 변수에 붙이든 관계없음

```
int *int_ptr;

int* int_ptr;
```

예 제

[예제 01] 문자열 출력 프로그램

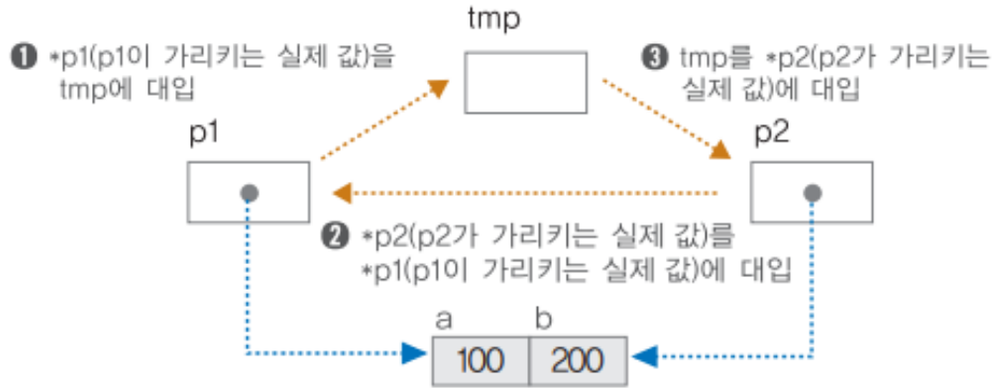
사용자로부터 최대 10개의 문자열들을 입력 받아 저장하고, 입력 받은 문자열들을 역순으로 출력하는 프로그램을 작성하시요.

1. 사용자가 “end”를 입력하면 입력 받는 것을 중단하고, 지금까지 입력 받은 문자열들을 역순으로 출력한다. (“end”는 제외)
2. 문자열의 최대 길이는 10을 넘지 않는다.
3. 사용자가 입력한 문자열은 스택 구조의 배열에 저장한다.

[예제 02] 포인터를 이용한 두 값의 교환

**사용자로부터 2개의 정수를 입력 받고,
포인터를 이용하여 두 개의 값을 교환한다음 출력 하시오.**

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
```

```
int a, b, tmp;
int *p1, *p2;
```

—— 정수형 변수 3개와 포인터 변수 2개를 선언한다.

Q & A