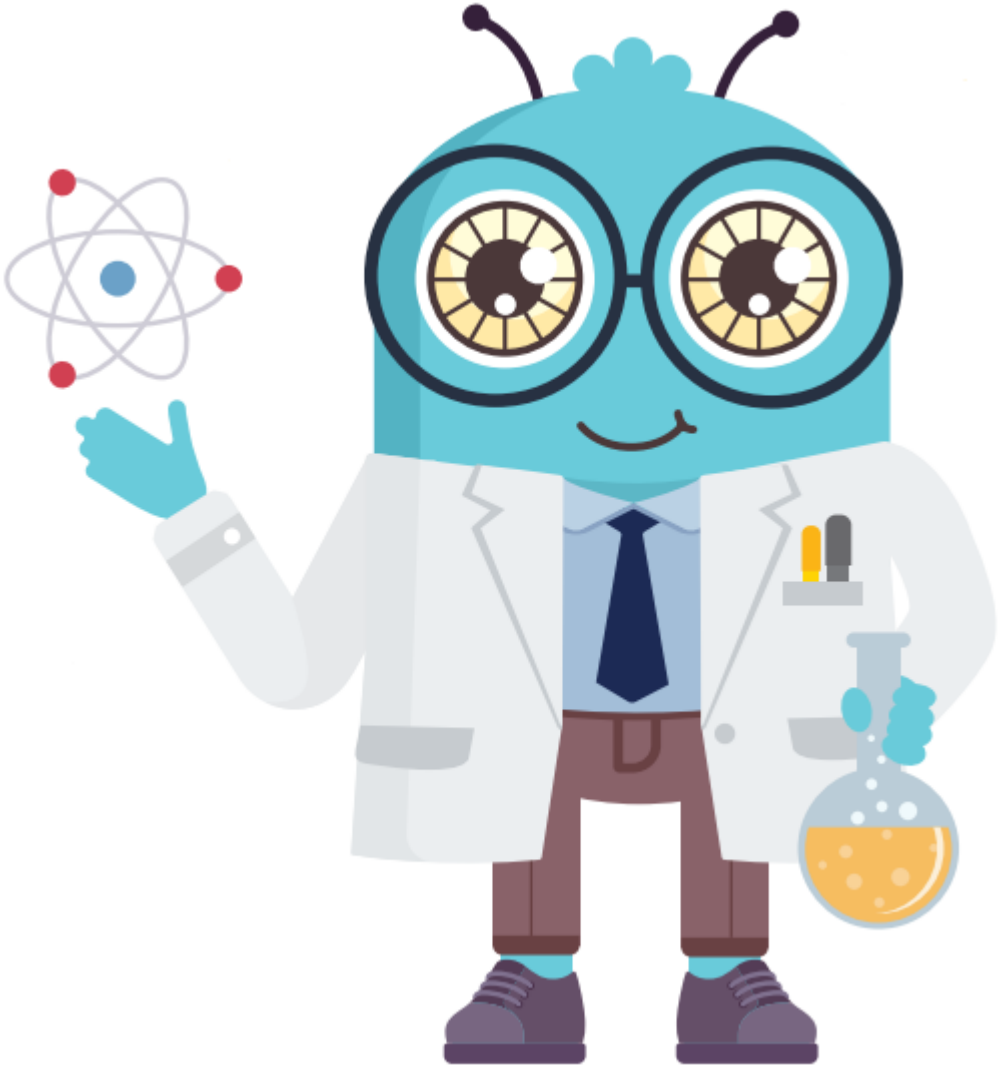

Chapter 12

고급 포인터

목차

01
메모리 할당 함수

1. 메모리 할당 함수

1. 꼭 기억해야 할 포인터의 주요 내용

▪ 메모리와 관련된 포인터의 기본 내용

- ❶ 컴퓨터의 모든 메모리에는 주소(Address)가 지정되어 있음([그림 9-6] 참조)
- ❷ int aa[3];과 같이 배열을 선언하면 배열 aa는 변수가 아닌 메모리의 주솟값 그 자체를 의미하고, 이를 '포인터 상수'라고도 함([그림 9-8] 참조)
- ❸ 포인터 변수란 "주소를 담는 그릇(변수)"이고, 포인터 변수를 선언할 때에는 int *p; 또는 char *p;와 같이 '*'를 붙여서 선언([그림 9-9] 참조)
- ❹ 포인터 변수에는 주소만 대입해야 하는데, 이는 변수 앞에 '&'를 붙이면 됨 ([그림 9-12], [그림 9-13] 참조)

1. 메모리 할당 함수

1. 꼭 기억해야 할 포인터의 주요 내용

- 메모리와 관련된 포인터의 기본 내용

기본 12-1 포인터를 사용하여 정수 합계를 구하는 예 12-1.c

```
01 #define  _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03 void main( )
04 {
05    int aa[3];                  ----- 정수형 배열을 선언한다.
06    int  *p;                    ----- 정수형 포인터 변수를 선언한다.
07    int i, hap=0;
08
09    for(i=0; i < 3; i++)        ----- 배열에 숫자 3개를 입력한다.
10    {
11       printf(" %d 번째 숫자 : ", i+1);
12       scanf("%d", &aa[i]);
13    }
14
```

1. 꼭 기억해야 할 포인터의 주요 내용

- 메모리와 관련된 포인터의 기본 내용

```
15    p = aa;                          ----- 포인터 변수에 배열 aa의 주소를 대입한다.
16
17    for(i=0; i < 3; i++)             ----- 합계를 누적한다. aa[0]~aa[2]의 합계를 구한다.
18      hap = hap + *(p+i);
19
20    printf("입력 숫자의 합=> %d\n", hap);
21 }
```

실행 결과

```
1 번째 숫자 : 10
2 번째 숫자 : 20
3 번째 숫자 : 30
입력 숫자의 합=> 60
```

1. 메모리 할당 함수

1. 꼭 기억해야 할 포인터의 주요 내용

- 메모리와 관련된 포인터의 기본 내용

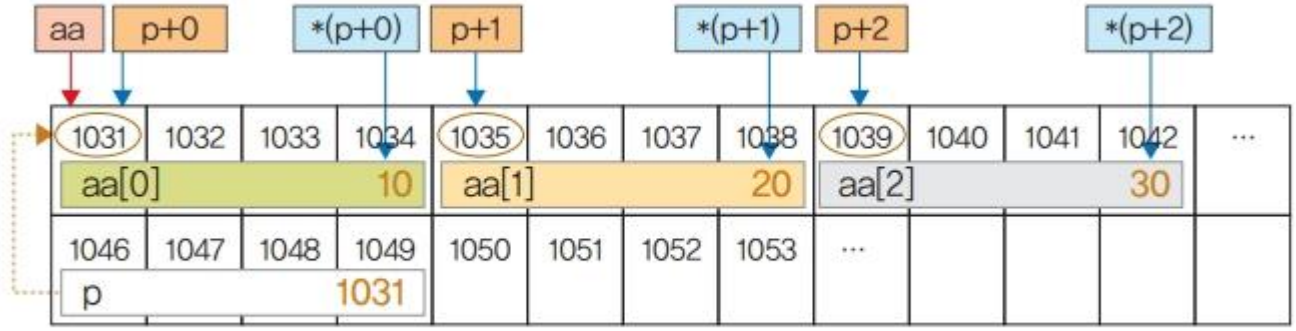

그림 12-1 배열과 포인터의 관계

- 5행에서 정수형 배열 aa[3]을 선언하면 1031~1042번지에 4바이트×3=12바이트의 메모리 확보
- 배열 aa는 1031번지 그 자체를 의미하는 포인터 상수

2. 동적 메모리 확보 : malloc( ) 함수

- **동적 메모리 확보의 개념**
  - 프로그램 실행 시 필요한 메모리 크기가 고정되는 경우 → 문제 없음
  - 필요로 하는 메모리의 크기가 다른 경우 → 메모리 낭비 문제 발생
  - 해결 방법 : 메모리를 미리 잡아두지 않고, 필요할 때마다 확보

    malloc() 함수 사용

1. 메모리 할당 함수

2. 동적 메모리 확보 : malloc( ) 함수

기본 12-2 고정된 크기의 배열로 인한 메모리 낭비의 예 12-2.c

```
01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03 void main( )
04 {
05    int aa[10000];          ---- 정수형 배열을 선언한다.
06    int  *p;                ---- 정수형 포인터 변수를 선언한다.
07    int i, hap=0;
08    int cnt;
09
10    printf(" 입력할 개수는 ? ");
11    scanf("%d", &cnt);      ---- 입력할 숫자의 개수를 입력한다.
12
13    for(i=0; i < cnt; i++)  ---- 입력한 개수만큼 배열에 숫자를 입력한다.
14    {
15       printf(" %d 번째 숫자 : ", i+1);
16       scanf("%d", &aa[i]);
17    }
```

1. 메모리 할당 함수

2. 동적 메모리 확보 : malloc( ) 함수

```
18
19    p = aa;                          ----- 포인터 변수에 주소를 대입한다.
20
21    for(i=0; i < cnt; i++)           ----- 합계를 누적한다.
22      hap = hap + *(p+i);
23
24    printf("입력 숫자의 합 ==> %d\n", hap);  ----- 합계를 출력한다.
25 }
```

실행 결과

```
입력할 개수는 ? 3
1 번째 숫자 : 10
2 번째 숫자 : 20
3 번째 숫자 : 30
입력 숫자의 합 ==> 60
```

- 겉으로는 이상이 없어 보이지만 실제로는 다음과 같은 문제가 있을 수 있음
  - 만약 10000개가 넘는 숫자를 더하고 싶을 때는 이 프로그램으로 계산할 수 없음
  - 전체 배열 중 3개만 사용하고 종료했으므로 사용되지 않은 나머지 9997개의 메모리는 낭비

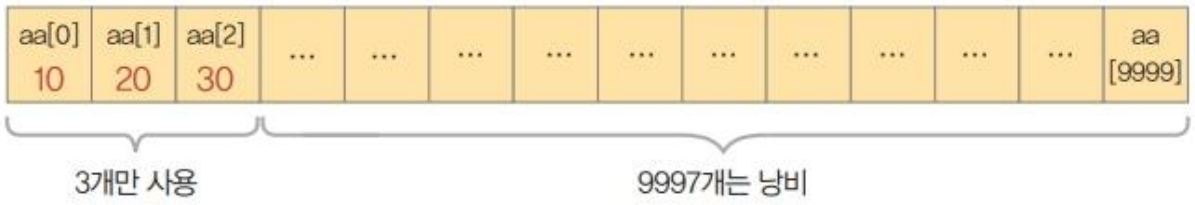

그림 12-2 과도한 메모리 낭비

1. 메모리 할당 함수

2. 동적 메모리 확보 : malloc( ) 함수

- 메모리의 낭비를 막으려면 [그림 12-3]과 같이 필요한 만큼의 메모리를 확보해서 사용

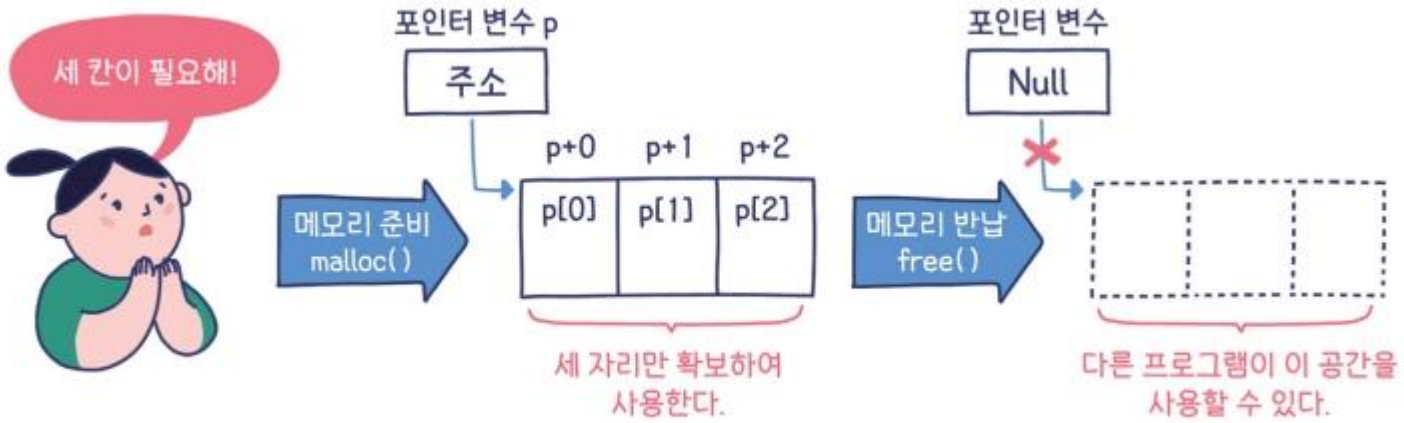

그림 12-3 동적 메모리 할당의 개념

- 먼저 더하려는 숫자가 몇 개인지 사용자에게 물어보고 malloc( ) 함수를 사용하여 메모리 확보
- 만약 사용자가 숫자 3개를 입력하고자 한다면 메모리 세 칸을 확보하고 확보한 주소를 포인터 변수에 넣음

1. 메모리 할당 함수

- **malloc( ) 함수의 사용 형식**

```
포인터 변수 = (포인터 변수의 데이터형*) malloc(포인터 변수의 데이터형 크기 × 필요한 크기)
```

- **malloc( ) 함수의 사용 예**
  - 포인터 변수를 int* p;로 선언한 경우

```
p = (int*) malloc(4 * 3);
```

  - int 형의 크기를 모를 경우, sizeof() 함수를 사용

```
p = (int*) malloc(sizeof(int) * 3);
```

- **malloc() 함수 사용 종료**
  - free() 함수 : 사용한 메모리를 반납
  - 포인터 변수에 널(null) 값을 넣는다는 의미
  - 포인터 변수는 아무것도 가리키지 않으므로, 이 공간을 운영체제에 반납

1. 메모리 할당 함수

- **malloc( ) 함수의 활용**

응용 12-3 malloc( ) 함수 사용 예 — 12-3.c

```
01 #define  _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03 #include <malloc.h>          ---- 메모리 관련 함수를 사용할 때 malloc.h를 추가해야 한다.
04 void main( )
05 {
06    int* p;                   ---- 정수형 포인터를 선언한다.
07    int i, hap=0;
08    int cnt;
09
10    printf(" 입력할 개수는 ? ");
11    scanf("%d", &cnt);        ---- 입력할 숫자의 개수를 입력한다.
12
13    p = (int*) __1__(sizeof(int) * cnt);   ---- 입력한 개수만큼 메모리를 확보한다.
14
15    for(i=0; i < cnt; i++)    ---- 입력한 개수(cnt)만큼 반복한다.
16    {
17       printf(" %d 번째 숫자 : ", i+1);
18       scanf("%d", __2__ );   ---- 공간이 확보된 포인터 변수 p에 입력받은 숫자를 입력한다. 배열처럼 &p[i]라고 입력해도 된다.
19    }
```

1. 메모리 할당 함수

- **malloc( ) 함수의 활용**

```
20
21    for(i=0; i < cnt; i++)
22       hap = hap + __3__          ----- 메모리의 실제 값을 합계에 누적한다. 배열처럼 p[i]라고 입력해도 된다.
23
24    printf("입력 숫자 합 ==> %d\n", hap);   ----- 합계를 출력한다.
25
26    free(p);                      ----- 메모리를 해제한다.
27 }
```

정답_ 1 malloc 2 p+i 3 *(p+i)

실행 결과

```
입력할 개수는 ? 3
1 번째 숫자 : 10
2 번째 숫자 : 30
3 번째 숫자 : 50
입력 숫자 합 ==> 90
```

1. 메모리 할당 함수

- **그 외 메모리 관련 함수의 활용**
  - **calloc()** : 처음부터 0으로 초기화된 메모리를 확보
    사용 형식은 malloc() 함수와 동일

기본 12-4 malloc( ) 함수와 calloc( ) 함수의 비교 — 12-4.c

```
01 #include <stdio.h>
02 #include <malloc.h>          ---- 메모리 관련 함수를 사용하기 위해 필요하다.
03
04 void main( )
05 {
06    int *p, *s;               ---- 정수형 포인터 변수를 선언한다.
07    int i, j;
08
09    printf("malloc( ) 함수 사용\n");
10    p = (int*) malloc(sizeof(int)*3);   ---- malloc( ) 함수로 정수형 메모리 3개를 할당한다
11                                            (3×4바이트=12바이트).
12    for(i=0; i < 3; i++)
13       printf("할당된 곳의 초깃값 p[%d] ==> %d\n", i, p[i]);   ---- 포인터 변수 p가 가리키는
14                                                                  곳의 실제 값을 출력한다.
15    free(p);                  ---- 메모리를 해제한다.
```

1. 메모리 할당 함수

- **그 외 메모리 관련 함수의 활용**

```
16
17    printf("\ncalloc( ) 함수 사용\n");
18    s = (int*) calloc(sizeof(int),3);
19
20    for(j=0; j < 3; j++)
21       printf("할당된 곳의 초깃값 s[%d] ==> %d\n", j, s[j]);
22
23    free(s);
24 }
```

calloc(크기, 개수) 함수로 정수형 메모리 3개를 할당한다.

포인터 변수 s가 가리키는 곳의 실제 값을 출력한다.

메모리를 해제한다.

실행 결과

```
malloc() 함수 사용
할당된 곳의 초깃값 p[0] ==> -842150451
할당된 곳의 초깃값 p[1] ==> -842150451
할당된 곳의 초깃값 p[2] ==> -842150451

calloc() 함수 사용
할당된 곳의 초깃값 s[0] ==> 0
할당된 곳의 초깃값 s[1] ==> 0
할당된 곳의 초깃값 s[2] ==> 0
```

1. 메모리 할당 함수

- **그 외 메모리 관련 함수의 활용**
  - **realloc()** : 메모리의 크기를 실시간으로 변경하는 함수

```
포인터 변수 = (포인터 변수의 데이터형*) realloc(기본 포인터, 포인터 변수의 데이터형 크기 × 필요한 크기);
```

예) p의 크기(개수)를 10으로 변경

```
p = (int*) realloc(p, sizeof(int) * 10);
```

  - 필요에 따라서 메모리의 크기를 실시간으로 변경하는 프로그램

응용 12-5 realloc( ) 함수 사용 예 — 12-5.c

```
01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03 #include <malloc.h>          ----- 메모리 관련 함수를 사용하기 위해 필요하다.
04 void main( )
05 {
06    int* p;
07    int i, hap=0;
08    int cnt=0;
09    int data;
10
```

1. 메모리 할당 함수

- 그 외 메모리 관련 함수의 활용

```
11    p = (int*) malloc(sizeof(int) * 1);
12    printf(" 1 번째 숫자 : ");
13    scanf("%d", &p[0]);                    ----- 첫 번째 값을 입력받고 데이터 개수를
14    cnt++;                                       1 증가시킨다.
15
16    for(i=2;  ; i++)                       ----- 두 번째 값부터 계속 입력받는다.
17    {                                            조건이 없으므로 무한 루프가 된다.
18      printf(" %d 번째 숫자 : ", i);
19      scanf("%d", &data);                  ------ 바로 위에서 입력된 값을 임시 장소에
20                                                  저장한다.
21      if(data != 0)                        ------ 입력된 값이 0이 아니면 메모리를 한 칸
22        p = (int*) __1__(p, sizeof(int) * i);     추가하고, 0이면 for문을 빠져나간다.
23      else
24        __2__
25
26      p[i-1] = data;                       ------ 추가한 메모리 공간에 임시 장소의 값을
27      cnt++;                                      대입하고 입력값의 개수를 1 증가시킨다.
28    }
29
30    for(i=0; i < cnt; i++)                 ----- 사용자가 입력한 개수(cnt)만큼
31      hap = hap + p[i];                          반복해서 합계를 구한다.
32
```

1. 메모리 할당 함수

- 그 외 메모리 관련 함수의 활용

```
33    printf("입력 숫자 합 ==> %d\n", hap);        ----- 합계를 출력한다.
34
35    free(p);                                    ----- 메모리를 해제한다.
36 }
```

정답_1 realloc 2 break;

실행 결과

```
1 번째 숫자 : 22
2 번째 숫자 : 45
3 번째 숫자 : 77
4 번째 숫자 : 0
입력 숫자 합 ==> 144
```

1. 메모리 할당 함수

- **그 외 메모리 관련 함수의 활용**
  - 처음에 malloc( ) 함수로 메모리 한 칸을 확보하고 사용자가 입력한 값을 넣음
  - 이때 입력한 값이 0이 아니면 realloc( ) 함수를 사용하여 크기를 늘려감
  - 사용자가 0을 입력하면 필요한 작업(여기서는 값을 모두 합 하는 작업)을 한 후 free( ) 함수를 사용하여 메모리를 해제

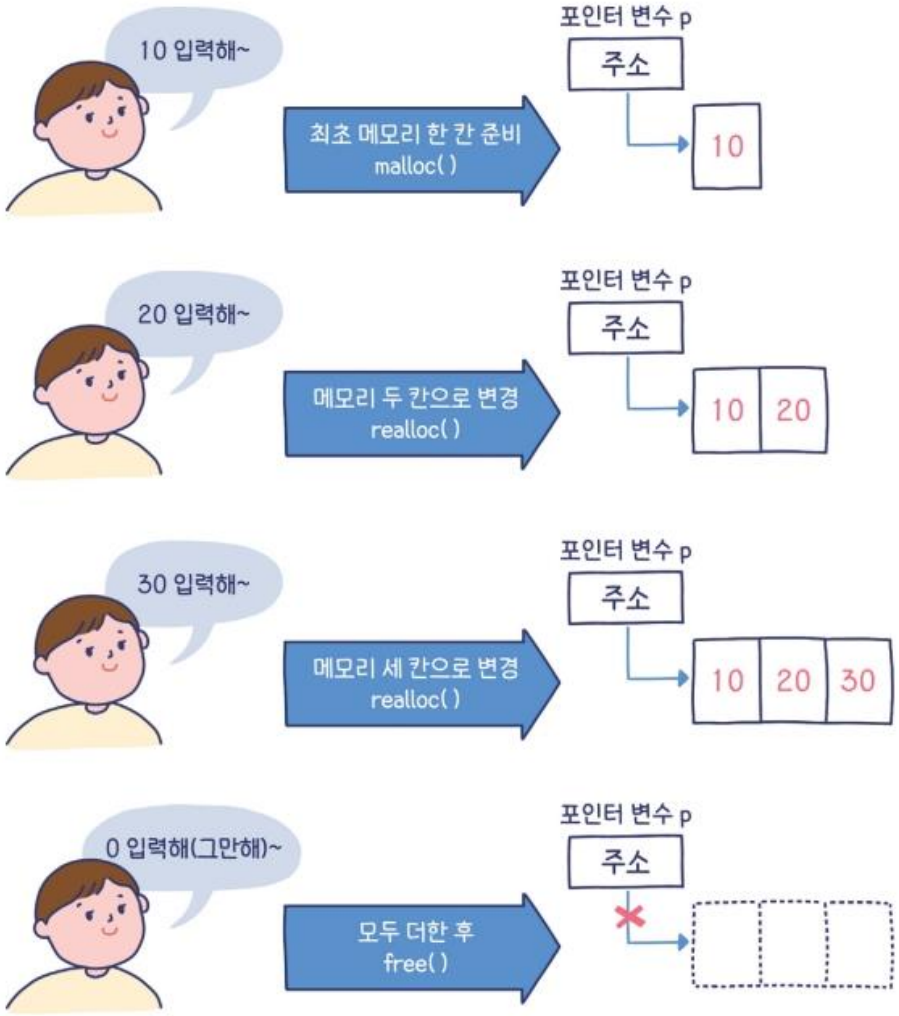

그림 12-4 realloc( ) 함수의 개념

02
포인터 배열

2. 포인터 배열

1. 여러 줄의 문자열을 처리 : 2차원 배열

- 문자 하나만 저장 : char
- 한 줄의 문자열 저장 : 배열 또는 포인터 변수 사용
- 여러 줄의 문자열을 저장 : 다차원 배열

기본 12-6 2차원 배열 사용 예 12-6.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    char data[3][100];          ----- 3행 100열의 2차원 배열을 선언한다.
06    int i;
07
08    for(i=0; i < 3; i++)        ----- 세 번 반복한다.
09    {
10       printf(" %d 번째 문자열 : ", i+1);
```

2. 포인터 배열

1. 여러 줄의 문자열을 처리 : 2차원 배열

```
11      gets(data[i]);
12    }
13
14    printf("\n — 입력과 반대로 출력(이차원 배열) —\n");
15    for(i=2; i >= 0; i—)
16    {
17      printf(" %d :%s\n", i+1, data[i]);
18    }
19  }
```

각 행에 최대 99자의 문자열을 입력한다.

2행, 1행, 0행의 순서로 마지막 행부터 출력한다.

실행 결과

```
1 번째 문자열 : Basic-C
2 번째 문자열 : Programming
3 번째 문자열 : Study

-- 입력과 반대로 출력(이차원 배열) --
3 :Study
2 :Programming
1 :Basic-C
```

2. 포인터 배열

1. 여러 줄의 문자열을 처리 : 2차원 배열

- 15~18행에서는 가장 최근에 입력된 3행부터 출력
- 즉 data[2], data[1], data[0]의 순서로 출력

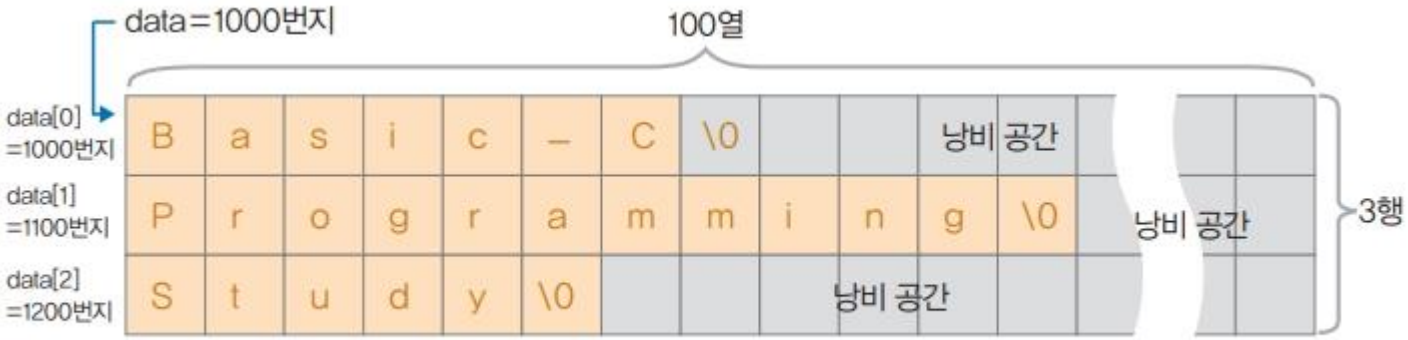

그림 12-5 2차원 배열의 메모리 낭비

- [그림 12-5]에서 보듯이 사용자가 입력한 글자가 100자가 되지 않으면 낭비되는 공간이 너무 많음
- 포인터 배열은 이런 공간 낭비의 단점을 극복하기 위한 것

2. 포인터 배열의 활용

- **포인터 배열 char* p[3] 선언**
  - 일반 배열처럼 p[0], p[1], p[2] 생성, 그 안에 주소 저장

그림 12-6 포인터 배열의 사용

- **일반 배열과 포인터 배열의 차이**
  - 일반 배열 : 정수 또는 문자가 들어감
  - 포인터 배열 : 주솟값이 들어감 (주솟값(예로 1000번지) 자체가 의미 있는 것이 아니라 그 주솟값이 가리키는 곳의 값이 중요함)

2. 포인터 배열의 활용

그림 12-7 일반 배열과 포인터 배열의 비교

2. 포인터 배열의 활용

응용 12-7 포인터 배열 사용 예 12-7.c

```
01 #include <stdio.h>
02 #include <malloc.h>          ----- 메모리 관련 함수와 문자열 관련 함수를 사용하기 위해
03 #include <string.h>                 필요하다.
04
05 void main( )
06 {
07    char* p[3];               ----- 세 칸의 포인터 배열을 선언한다.
08    char  imsi[100];          ----- 입력값을 저장할 임시 공간 배열이다.
09    int i, size;
10
11    for(i=0; i < 3; i++)
12    {
```

2. 포인터 배열의 활용

```
13      printf(" %d 번째 문자열 : ", i+1);
14      gets(imsi);                          ----- 임시 공간에 문자열을 입력한다.
15
16      size = strlen(imsi);                 ----- 입력한 문자열의 길이를 계산한다.
17      p[i]  = (char*) malloc((sizeof(char) * size) + 1);   -----'입력한 길이+1' 크기의
18                                                            메모리를 확보한다.
19      strcpy(__1__, imsi);                 ----- 입력한 문자열(imsi)의 내용을 메모리를 확보한
20    }                                              공간에 복사한다.
21
22    printf("\n — 입력과 반대로 출력(포인터) —\n");
23    for(i=2; i >= 0; i—)
24    {
25      printf(" %d :%s\n", i+1, p[i]);      ----- 포인터 배열에 저장된 문자열을 출력한다.
26    }
27
28    for(i=0; i < 3; i++)                   ----- 할당했던 메모리 3개를 운영체제에 반납한다.
29      free(p[i]);
30  }
```

2. 포인터 배열의 활용

실행 결과

```
1 번째 문자열 : Basic-C
2 번째 문자열 : Programming
3 번째 문자열 : Study

-- 입력과 반대로 출력(포인터) --
3 :Study
2 :Programming
1 :Basic-C
```

*

예제 모음

[예제모음 32] 여러 숫자 중 짝수만 더하기

예제 설명 사용자가 입력한 여러 숫자 중에서 짝수의 합계를 출력하는 프로그램이다([응용 12-3] 활용).

실행 결과

```
입력할 개수는 ? 4
1 번째 숫자 : 2
2 번째 숫자 : 40
3 번째 숫자 : 7
4 번째 숫자 : 11
입력한 짝수합 ==> 42
```

[예제모음 32] 여러 숫자 중 짝수만 더하기

```
01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03 #include <malloc.h>
04 void main( )
05 {
06    int* p;                                  ----- 정수형 포인터 변수를 선언한다.
07    int i, hap=0;
08    int cnt;
09
10    printf(" 입력할 개수는 ? ");
11    scanf("%d", &cnt);                       ----- 사용자가 입력할 숫자의 개수를 입력한다.
12    p = (int*) malloc(sizeof(int) * cnt);    ----- 입력한 개수(cnt)에 따라 메모리를 확보한다.
```

[예제모음 32] 여러 숫자 중 짝수만 더하기

```
13
14    for(i=0; i < cnt; i++)                  ----- cnt만큼 배열에 숫자를 입력한다.
15    {
16       printf(" %d 번째 숫자 : ", i+1);
17       scanf("%d", p+i);
18    }
19
20    for(i=0; i < cnt; i++)
21    {
22       if(p[i] % 2 == 0)                    ----- 짝수일 때만 값을 누적한다.
23          hap = hap + p[i];
24    }
25
26    printf("입력한 짝수합 ==> %d\n", hap);  ----- 합계를 출력한다.
27
28    free(p);                                ----- 메모리를 해제한다.
29 }
```

[예제모음 33] 입력한 문자열을 반대 순서로 출력

예제 설명 입력한 순서의 반대로 그리고 각 행의 문자도 반대 순서로 출력하는 프로그램이다([응용 12-7] 활용).

실행 결과

```
1 번째 문자열 : IT CookBook
2 번째 문자열 : Basic C
3 번째 문자열 : Programming

-- 입력과 반대로 출력(포인터) : 글자 순서도 거꾸로 --
3 :gnimmargorP
2 :C cisaB
1 :kooBkooC TI
```

[예제모음 33] 입력한 문자열을 반대 순서로 출력

```
01 #include <stdio.h>
02 #include <malloc.h>
03 #include <string.h>
04
05 void main( )
06 {
07    char* p[3];                          ---- 포인터 배열을 세 칸 선언한다.
08    char  imsi[100];                     ---- 입력값을 저장할 임시 공간을 마련한다.
09    int i, k, size;
10
11    for(i=0; i < 3; i++)                 ---- 20행까지의 동작을 세 번 반복한다.
12    {
13       printf(" %d 번째 문자열 : ", i+1);
14       gets(imsi);                       ---- 임시 공간에 문자열을 입력한다.
15
16       size = strlen(imsi);              ---- 입력한 문자열의 길이를 계산한다.
17       p[i] = (char*) malloc( (sizeof(char) * size) + 1 );  ---- '입력한 길이+1' 크기의
18                                                              메모리를 확보한다.
19       strcpy(p[i], imsi);               ---- 확보된 메모리에 입력한 문자열을 복사한다.
20    }
21
22    printf("\n -- 입력과 반대로 출력(포인터) : 글자 순서도 거꾸로 --\n");
23    for(i=2; i >= 0; i--)                ---- 31행까지의 동작을 세 번 반복한다.
24    {
```

[예제모음 33] 입력한 문자열을 반대 순서로 출력

```
25        size = strlen(p[i]);          ----- 문자열의 길이를 체크한다.
26        imsi[size] = '\0';            ----- 문자열의 끝부분에 널 문자를 입력한다.
27        for(k=size-1; k >= 0; k--)    ----- '문자열 길이-1'만큼 반복하며 p[i]와 imsi의
28          imsi[size-1-k] = p[i][k];         문자열 위치를 바꾼다.
29
30        printf(" %d :%s\n", i+1, imsi);  ----- imsi 배열에 저장된 문자열을 출력한다.
31    }
32
33    for(i=0; i < 3; i++)              ----- 할당했던 메모리 3개를 운영체제에 반납한다.
34        free(p[i]);
35  }
```

감사합니다!