

# 컴퓨터 프로그래밍2

## 동적할당

---

장서윤 [pineai@cnu.ac.kr](mailto:pineai@cnu.ac.kr)

# 동적 할당 함수

---

## ❖ 동적 할당 - 실행 시점에 메모리 공간 할당

표 16-1 동적 할당 관련 함수

함수	구분	사용 예
malloc	원형	<code>void *malloc(unsigned int size);</code>
	기능	size 바이트 수 만큼 할당하고 시작 위치 반환
	사용 예	<code>int *p = (int *) malloc(sizeof(int));</code>
calloc	원형	<code>void *calloc(unsigned int cnt, unsigned int size);</code>
	기능	(cnt * size) 바이트 수 만큼 할당하고 0으로 초기화 후 시작 위치 반환
	사용 예	<code>double *p = (double *) calloc(5 * sizeof(double));</code>
realloc	원형	<code>void *realloc(void *p, unsigned int size);</code>
	기능	p가 연결한 영역의 크기를 size 바이트의 크기로 조정하고 시작 위치 반환
	사용 예	<code>char *p = (char *) realloc(p, 2 * strlen(str));</code>
free	원형	<code>void free(void *p);</code>
	기능	p가 연결한 영역 반환

# 동적 할당 함수

---

- ▶ 프로그램에 필요한 저장 공간
  - ▶ 프로그램 작성할 때 변수나 배열 선언 통해 확보
    - ▶ 프로그램의 실행 중에 저장 공간 할당할 수도
- ▶ 사용한 저장 공간은 재사용 위해 다시 반납
- ▶ 메모리 동적 할당할 때는 malloc 함수
- ▶ 반환할 때는 free 함수 사용

# MALLOC, FREE 함수

---


- ▶ 프로그램에 필요한 저장 공간
  - ▶ Malloc, free 함수 사용할 때 - **stdlib.h** 헤더 파일 인클루드
  - ▶ 두 함수의 원형

---

```
void *malloc(unsigned int size);  
void free(void *p);
```

---

## 예제 16-1 동적 할당한 저장 공간을 사용하는 프로그램

```
1. #include <stdio.h>
2. #include <stdlib.h>           // malloc, free 함수 사용을 위한 헤더 파일
3.                               .....
4. int main(void)
5. {
6.     int *pi;                  // 동적 할당 영역을 연결할 포인터 선언
7.     double *pd;
8.
9.     pi = (int *) malloc(sizeof(int));    // 메모리 동적 할당 후 포인터 연결
10.    if(pi == NULL)                    // 동적 할당에 실패하면 NULL 포인터 반환
11.    {
12.        printf("#으로 메모리가 부족합니다.\n"); // 예외 상황 메시지 출력
13.        exit(1);  // 프로그램 종료
14.    }
15.    pd = (double *) malloc(sizeof(double));
16.
17.    *pi = 10;                      // 포인터로 동적 할당 영역 사용
18.    *pd = 3.4;
19.
20.    printf("정수형으로 사용 : %d\n", *pi);    // 동적 할당 영역에 저장된 값 출력
21.    printf("실수형으로 사용 : %lf\n", *pd);
22.
23.    free(pi);                      // 동적 할당 영역 반환
24.    free(pd);
25.
26.    return 0;
27. }
```

실행  
결과 정수형으로 사용 : 10  
실수형으로 사용 : 3.4

# MALLOC, FREE 함수

---

- ▶ 메모리 동적 할당

- ▶ int형 변수로 사용하기 위해서는 4바이트, double형 변수로 사용하기 위해서는 8바이트 할당

- ▶ 필요한 바이트 수 malloc 함수의 인수로 줄 것

- ▶ 각 자료형에 대한 크기를 계산하여 주는 것이 좋음

- ▶ 컴파일러에 따라 int형 변수의 크기가 다르더라도 프로그램 수정 필요 없음

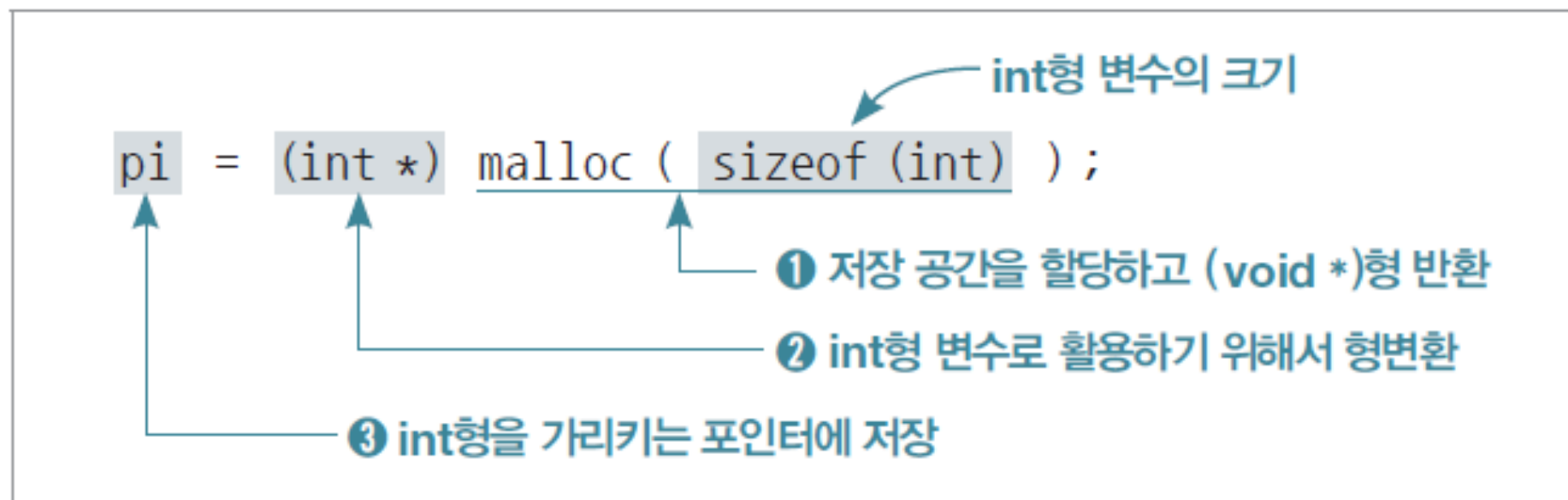
- ▶ 주어진 인수의 바이트 크기 만큼 메모리에서 연속된 저장 공간 할당 후 그 시작 주소 반환

# MALLOC, FREE 함수

---

▶ malloc 함수는 (void \*) 형 반환

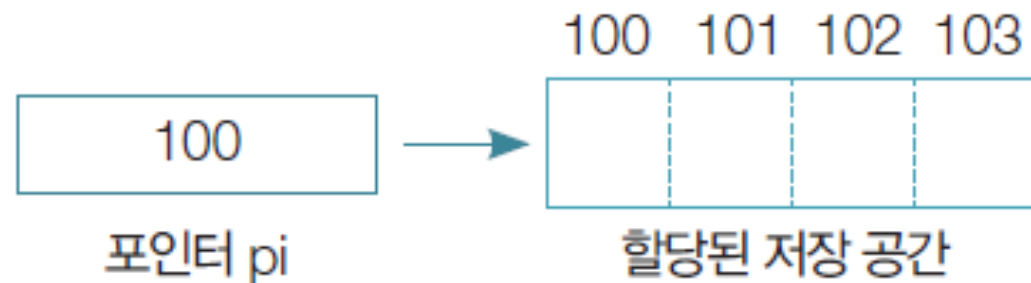
▶ 용도에 맞는 포인터형으로 형변환하여 사용



# MALLOC, FREE 함수

---

- ▶ 메모리 100번지부터 할당되었다 가정할 경우



- ▶ 반환값 포인터에 저장 후
  - ▶ 간접참조 연산 수행하여 가리키는 저장 공간 값을 저장하거나 출력 가능

---

```
*pi = 10;           // pi가 가리키는 저장 공간에 10 저장
printf("%d", *pi);  // pi가 가리키는 저장 공간의 값 출력
```

---

```
scanf("%d", &(*pi) = pi);
```



# MALLOC, FREE 함수

---

- ▶ 동적 메모리 사용시 주의점
  - ▶ malloc 함수 반환값이 널 포인터인지 반드시 확인 후 사용
    - ▶ 메모리 할당 함수는 원하는 크기의 공간 할당하지 못하면 0번지인 널 포인터(null pointer) 반환
    - ▶ 널 포인터는 보통 NULL로 표기하는데 전처리 단계에서 0으로 바뀌므로 정수0과 같다고 생각
    - ▶ 포인터의 특별한 상태를 나타내기 위해 사용하므로 간접 참조 연산불가
    - ▶ malloc 함수가 널 포인터 반환한 경우 그 값을 참조하면 실행 중 오류 메시지 표시하고 비정상 종료
    - ▶ 프로그램이 실행될 때 메모리의 상태에 따라 달라짐
    - ▶ 동적 할당 함수를 호출한 후에는 반드시 반환값을 검사하는 과정 필요

# MALLOC, FREE 함수

---

- ▶ 메모리 동적 할당할 때 사용이 끝난 저장 공간은 반환해야 함
- ▶ 자동 지역 변수의 저장 공간
  - ▶ 함수가 반환될 때 자동으로 회수
- ▶ 동적으로 할당한 저장공간
  - ▶ 함수가 반환된 후에도 자동으로 회수되지 않음
  - ▶ 반환되기 전에 free 함수로 직접 반환
    - ▶ main 함수가 끝날 때는 굳이 반환할 필요가 없음 (프로그램 끝나면 동적 할당 부분 자동 반납)

# MALLOC, FREE 함수

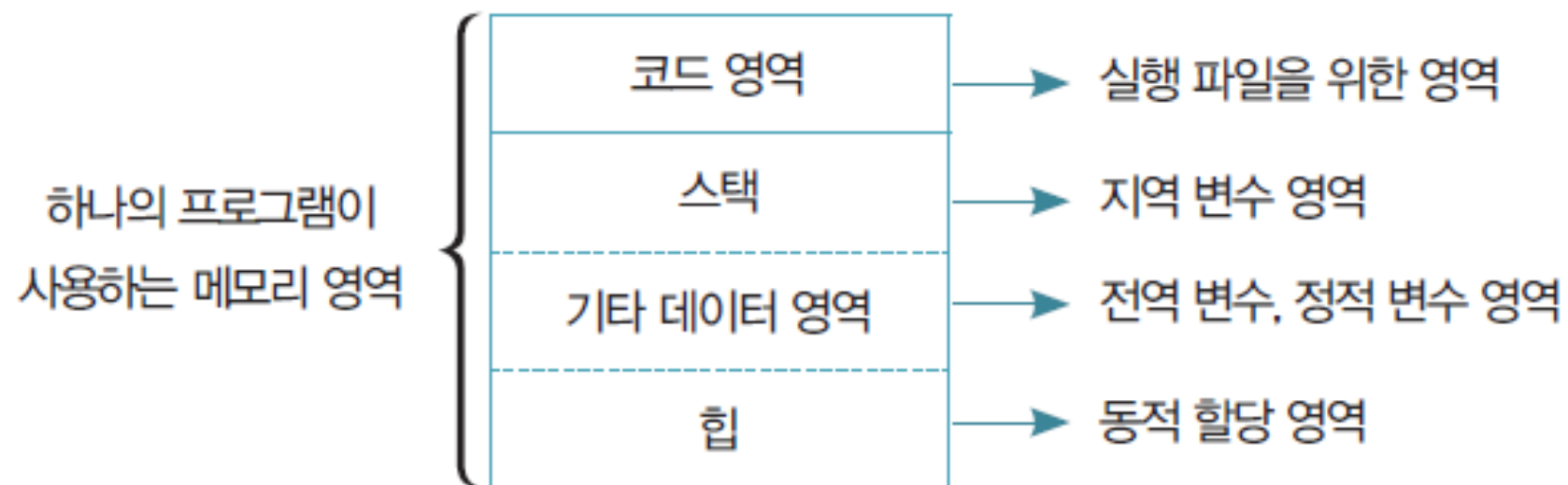
---

- ▶ 동적으로 할당한 저장공간
  - ▶ 그 외 다른 함수에서 사용하던 저장 공간은 불필요하면 반환
- ▶ 메모리 반환코드 생략해도 당장 프로그램 실행에 영향 미치지 않아 반환 코드 생략 시
  - ▶ 메모리 누수(memory leak) 발생 가능
- ▶ 프로그램 의도치 않게 종료 가능

# MALLOC, FREE 함수

---

- ▶ 프로그램이 사용하는 메모리 영역은 기억부류 가짐
  - ▶ 프로그램은 실행될 때 일정한 메모리 영역 사용
- ▶ 이 영역은 다시 몇 개의 영역으로 나뉘어 관리
  - ▶ 기억부류(storage class)



- ▶ 구체적인 구분은 시스템에 따라 다름

# MALLOC, FREE 함수

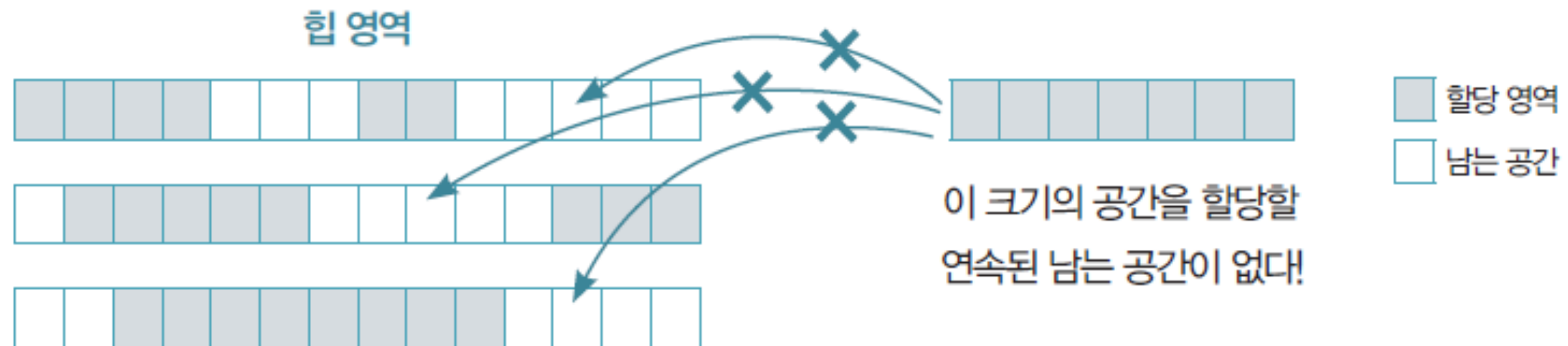
---

- ▶ 힙에 할당한 저장 공간
  - ▶ 지역 변수와 마찬가지로 쓰레기값 존재
  - ▶ 프로그램이 종료될 때까지 메모리에 존재
    - ▶ 주소만 알면 특정 함수에 구매 받지 않고 어디서나 사용 가능 지역 변수와 달리
- ▶ 동적 할당된 저장 공간
  - ▶ 함수가 반환되어도 메모리 회수되지 않음
  - ▶ 메모리에 저장 공간이 넉넉히 남아 있어도 널 포인터 반환 가능

# MALLOC, FREE 함수

---

- ▶ 힙 영역 - 메모리의 사용과 반환이 불규칙적
  - ▶ 사용 가능한 영역이 조각나서 흩어져 있을 수 있음
    - ▶ 연속된 큰 저장 공간 요구하면 malloc 함수는 원하는 저장 공간 찾지 못하고 널 포인터 반환



- ▶ 동적 할당 함수 호출한 후 반드시 반환값 검사하여 메모리의 할당 여부 확인해야

# 동적 할당 영역을 배열처럼 쓰기

---

- ▶ 형태가 같은 변수가 많이 필요할 때
  - ▶ 하나씩 동적 할당하는 것은 비효율적
    - ▶ 할당한 저장 공간 수만큼 포인터 필요
  - ▶ 많은 저장 공간 한꺼번에 동적 할당하여 배열처럼 사용
    - ▶ 할당한 저장 공간의 시작 위치만 포인터에 저장하면 포인터를 배열처럼 사용 가능

## 예제 16-2 동적 할당 영역을 배열처럼 사용

```
1. #include <stdio.h>
2. #include <stdlib.h>
3.
4. int main(void)
5. {
6.     int *pi;                // 동적 할당 영역을 연결할 포인터
7.     int i, sum = 0;         // 반복 제어 변수와 누적 변수
8.
9.     pi = (int *) malloc(5 * sizeof(int)); // 저장 공간 20바이트 할당
10.    if(pi == NULL)
11.    {
12.        printf("메모리가 부족합니다!\n");
13.        exit(1);
14.    }
15.    printf("다섯 명의 나이를 입력하세요 : ");
16.    for(i = 0; i < 5; i++)    // i는 0부터 4까지 5번 반복
17.    {
18.        scanf("%d", &pi[i]); // 배열 요소에 입력
19.        sum += pi[i];         // 배열 요소의 값 누적
20.    }
21.    printf("다섯 명의 평균 나이 : %.1lf\n", (sum / 5.0)); // 평균 나이 출력
22.    free(pi);               // 할당한 메모리 영역 반환
23.
24.    return 0;
25. }
```

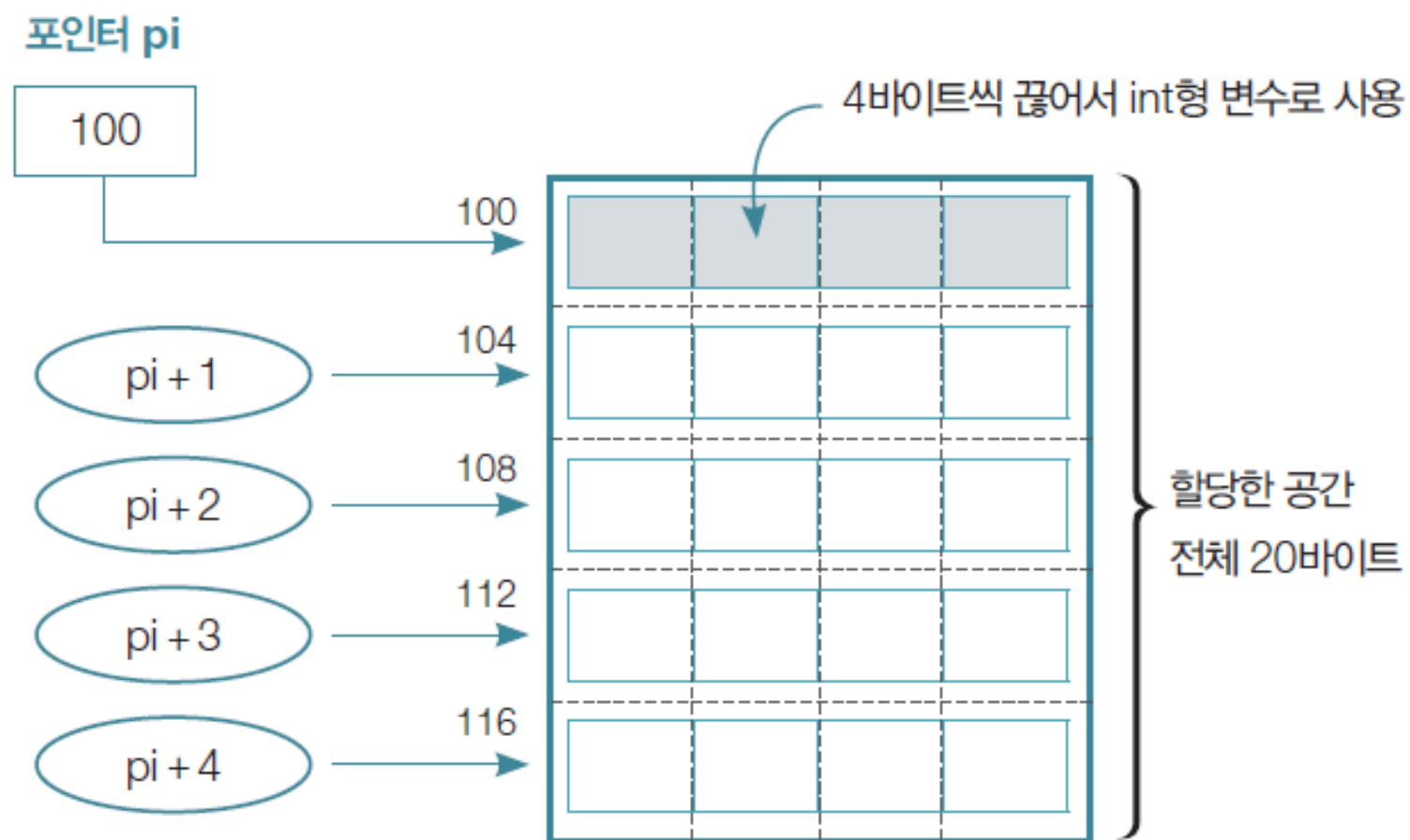
실행 결과 다섯 명의 나이를 입력하세요 : 21 27 24 22 35 ☐  
다섯 명의 평균 나이 : 25.8



# 동적 할당 영역을 배열처럼 쓰기

---

- ▶ 배열처럼 사용할 전체 저장 공간 동적 할당
  - ▶ int형 가리키는 포인터에 그 주소 저장
  - ▶ 포인터의 주소값을 int형의 크기만큼 증가
    - ▶ 전체 저장 공간 배열처럼 사용



# 기타 동적 할당 함수

---

## ➤ calloc 함수

➤ 메모리 동적 할당하여 0으로 초기화된 메모리 공간 얻음

➤ 함수 원형

---

```
void *calloc(unsigned int, unsigned int);
```

---

## ➤ realloc 함수

➤ 저장 공간 크기 조절

➤ 함수 원형

---

```
void *realloc(void *, unsigned int);
```

---

# 동적 할당 함수

---

## 예제 16-3 calloc, realloc 함수를 사용한 양수 입력

---

```
1. #include <stdio.h>
2. #include <stdlib.h>
3.
4. int main(void)
5. {
6.     int *pi           // 할당한 저장 공간을 연결할 포인터
7.     int size = 5;      // 한 번에 할당할 저장 공간의 크기, int형 변수 5개씩
8.     int cnt = 0;       // 현재 입력된 양수 개수
9.     int num;           // 양수를 입력할 변수
10.    int i;             // 반복 제어 변수
11.
12.    pi = (int *)calloc(size, sizeof(int)); // 먼저 5개의 저장 공간 할당
13.    while(1)
14.    {
```

```

15.     printf("양수를 입력하세요 => ");
16.     scanf("%d", &num);           // 데이터 입력
17.     if(num <= 0) break;           // 0또는 음수이면 종료 .....
18.     if(cnt < size)                 // 저장 공간이 남아 있으면
19.     {
20.         pi[cnt++] = num;           // 입력한 값 저장
21.     }
22.     else                           // 저장 공간이 부족하면
23.     {
24.         size += 5;                  // 크기를 늘려서 재할당
25.         pi = (int *)realloc(pi, size*sizeof(int));
26.         pi[cnt++] = num;           // 재할당한 공간에 값 대입
27.     }
28. }
30. for(i = 0; i < cnt; i++)
31. {
32.     printf("%5d", pi[i]);          // 입력한 데이터 출력
33. }
34. free(pi);                          // 동적 할당 저장 공간 반납
35.
36. return 0;
37. }

```

# 기타 동적 할당 함수

---

실행  
결과

양수를 입력하세요 => 1

양수를 입력하세요 => 2

양수를 입력하세요 => 3

양수를 입력하세요 => 4

양수를 입력하세요 => 5

양수를 입력하세요 => 6

양수를 입력하세요 => 7

양수를 입력하세요 => -1

1    2    3    4    5    6    7

# 기타 동적 할당 함수

---

- ▶ calloc 함수를 호출하는데 인수는 2개
  - ▶ 두 번째 인수는 malloc 함수와 마찬가지로 할당할 저장 공간의 크기 바이트 단위로 줌
- ▶ 첫 번째 인수로 그 개수
  - ▶ Ex) 배열 요소가 5개인 double형 배열처럼 사용할 공간 필요하다면 다음과 같이 사용

```
double *pd;
```

```
pd = ( double * ) calloc ( 5 , sizeof (double) );
```

배열 요소 수

double형 변수 하나의  
바이트 크기

# 기타 동적 할당 함수

---

- ▶ `calloc` 함수는 할당한 저장 공간을 모두 0으로 초기화
  - ▶ 0으로 초기화가 필요한 경우 따로 초기화 안 해도 됨.
- ▶ 저장 공간의 크기를 조정해야 하면 `realloc` 함수 사용
  - ▶ 이미 할당한 저장 공간의 포인터와 조정할 저장 공간의 전체 크기 줌

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

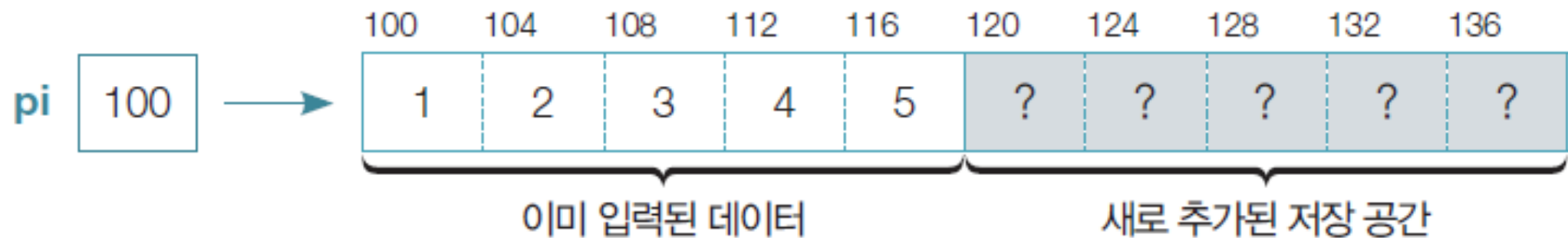
이미 할당한  
저장 공간의 포인터

재할당 후 전체  
저장 공간의 크기

# 기타 동적 할당 함수

---

- ▶ 저장 공간 늘리는 경우 이미 입력한 값은 그대로 유지
  - ▶ 추가된 공간에는 쓰레기값 존재
- ▶ 저장 공간 줄이는 경우
  - ▶ 입력된 데이터는 잘려나감
  - ▶ 저장 공간의 크기 조정 후 다시 그 주소를 반환하므로 포인터에 저장해 사용





# 기타 동적 할당 함수

---

- ▶ 이미 사용하던 저장 공간의 위치를 포인터가 기억
  - ▶ 재할당 과정에서 메모리의 위치 바뀔 수 있으므로 항상 `realloc` 함수가 반환하는 주소를 다시 포인터에 저장해 사용
- ▶ 메모리의 위치가 바뀌는 경우
  - ▶ 이미 있던 데이터는 계속 사용할 수 있도록 옮겨 저장
    - ▶ 사용하던 저장 공간은 자동 반환
  - ▶ 첫 번째 인수가 널 포인터인 경우
    - ▶ `malloc`과 같은 기능 수행
    - ▶ 두 번째 인수의 크기만큼 동적 할당 후 주소 반환

# 동적 할당 저장 공간의 활용

표 16-2 동적 할당 활용 방법

구분	설명	상세
입력 문자열 처리	사용 예	입력 문자열의 길이에 맞는 저장 공간 확보
	구현 방법	<pre>char str[80]; char *ps; ps = (char *) malloc(strlen(str) + 1); strcpy(ps, str);</pre>
명령행 인수 처리	main 함수의 인수	<pre>int main(int argc, char **argv);</pre>
	의미	argc - 명령행 문자열의 수, argv - 명령행의 문자열
	사용 예	<pre>for(i = 0; i &lt; argc; i++) {     printf("%s\n", argv[i]); }</pre>

#### 예제 16-4 3개의 문자열을 저장하기 위한 동적 할당

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4.
5. int main(void)
6. {
7.     char temp[80];                // 임시 char 배열
8.     char *str[3];                // 동적 할당 영역을 연결할 포인터 배열
9.     int i;                        // 반복 제어 변수
10.
11.     for(i = 0; i < 3; i++)
12.     {
13.         printf("문자열을 입력하세요 : ");
14.         gets(temp);              // 문자열 입력
15.         str[i] = (char *) malloc(strlen(temp) + 1); // 문자열 저장 공간 할당
16.         strcpy(str[i], temp);    // 동적 할당 영역에 문자열 복사
17.     }
18.
19.     for(i = 0; i < 3; i++)
20.     {
21.         printf("%s\n", str[i]);  // 입력된 문자열 출력
22.     }
23.
24.     for(i = 0; i < 3; i++)
25.     {
26.         free(str[i]);            // 동적 할당 영역 반환
27.     }
28.
29.     return 0;
30. }
```

실행  
결과

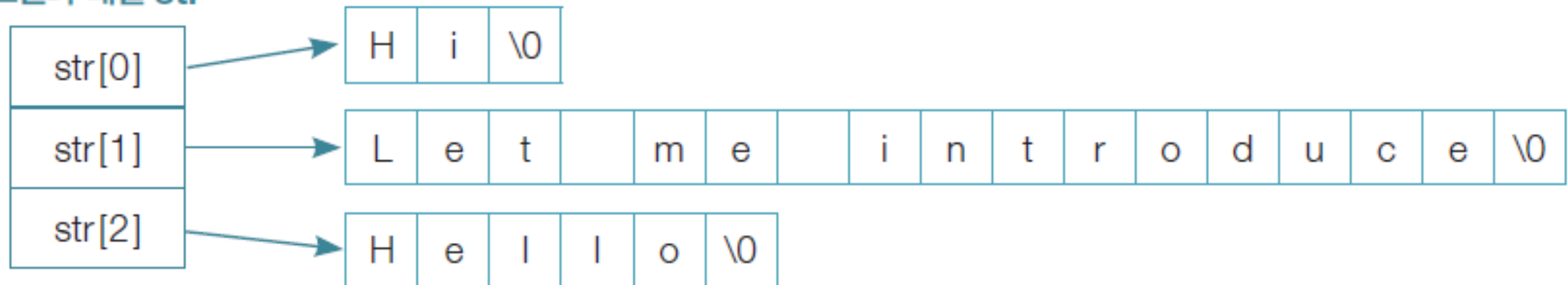
```
문자열을 입력하세요 : Hi
문자열을 입력하세요 : Let me introduce
문자열을 입력하세요 : Hello
Hi
Let me introduce
Hello
```

# 동적 할당을 사용한 문자열 처리

---

- ▶ 동적 할당 영역 연결할 포인터가 필요
  - ▶ 동적 할당 모두 끝낸 후의 상태

포인터 배열 `str`



- ▶ 입력한 문자열의 길이 계산 (malloc 함수의 인수)
  - ▶ strlen 함수는 널문자 제외 문자열 길이 계산
    - ▶ malloc 함수에 인수로 줄 때는 1 더해서 널문자도 포함할 수 있도록 저장 공간 할당
    - ▶ Malloc 함수가 반환하는 주소는 포인터 배열의 요소에 저장하여 할당한 저장 공간 연결
- ▶ 할당한 저장 공간에 입력한 문자열 복사 (포인터 배열)

## 예제 16-5 동적 할당 영역의 문자열을 함수로 출력

---

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4.
5. void print_str(char **ps);           // 동적 할당 영역의 문자열을 출력하는 함수
6.
7. int main(void)
8. {
9.     char temp[80];                   // 임시 char 배열
10.    char *str[21] = {0} ;            // 문자열을 연결할 포인터 배열, 널포인터로 초기화
11.    int i = 0;                        // 반복 제어 변수
12.
13.    while(i < 20)                     // 최대 20개까지 입력
14.    {
15.        printf("문자열을 입력하세요 : ");
16.        gets(temp);                  // 문자열 입력
17.        if(strcmp(temp, "end") == 0) break; // end가 입력되면 반복 종료
18.        str[i] = (char *) malloc(strlen(temp) + 1); // 문자열 저장 공간 할당
19.        strcpy(str[i], temp);         // 동적 할당 영역에 문자열 복사
20.        i++;
21.    }
22.    print_str(str);                   // 입력한 문자열 출력
23.
```

.....

```

24.   for(i = 0; str[i] != NULL; i++)      // str에 연결된 문자열이 없을 때까지
25.   {
26.       free(str[i]);                    // 동적 할당 영역 반환
27.   }
28.
29.   return 0;
30. }
31.
32. void print_str(char **ps)              // 2중 포인터 선언
33. {
34.     while(*ps != NULL)                 // 포인터 배열의 값이 널포인터가 아닌 동안 반복
35.     {
36.         printf("%s\n", *ps);           // ps가 가리키는 것은 포인터 배열의 요소
37.         ps++;                           // ps가 다음 배열 요소를 가리킨다.
38.     }
39. }

```

실행 결과

```

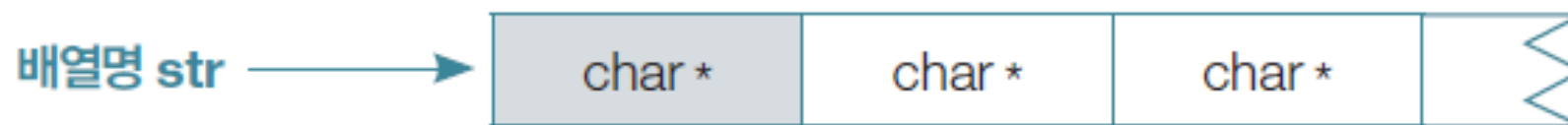
문자열을 입력하세요 : Hi
문자열을 입력하세요 : Let me introduce
문자열을 입력하세요 : Hello
문자열을 입력하세요 : end
Hi
Let me introduce
Hello

```

# 동적 할당 저장 공간의 활용

---

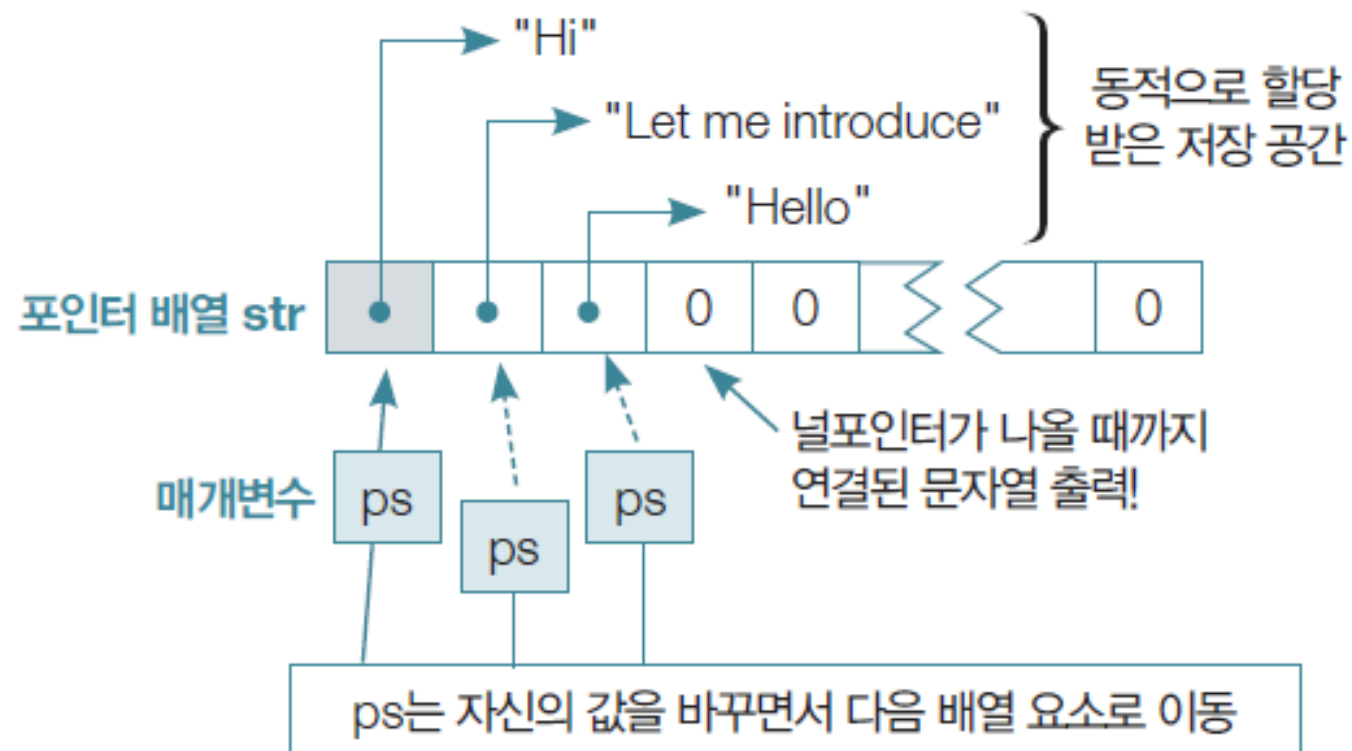
- ▶ char 배열의 문자열을 출력하는 함수
  - ▶ 배열명을 저장할 포인터를 매개변수로 선언
- ▶ 마찬가지로 포인터 배열의 문자열을 출력하는 함수도 포인터 배열의 이름을 저장할 포인터 매개 변수가 필요
- ▶ 10행의 배열명 str - 포인터 배열의 첫 번째 요소 가리키므로 가리키는 것의 형태는 (char \*) 형



# 동적 할당 저장 공간의 활용

---

- ▶ while문 사용하여 코드 작성
  - ▶ 배열명 포인터에 저장
    - ▶ 포인터 자신의 값 바꿀 수 있음
    - ▶ 매개변수 하나씩 증가시키면서 문자열 출력





# 동적 할당 저장 공간의 활용

---

- ▶ 포인터나 포인터 배열 자동 지역 변수로 선언
  - ▶ 쓰레기값이 주소로 존재
    - ▶ 쓰레기값이 참조가 불가능한 코드 영역의 주소일 때 부주의로 이 값을 참조한다면 프로그램은 중간에 실행을 멈춤
- ▶ 포인터 배열은 선언과 동시에 널 포인터로 초기화
- ▶ 참조할 때 널 포인터인지 검사

# 동적 할당 저장 공간의 활용

---

- ▶ 최소한 포인터 배열의 마지막 요소는 널 포인터의 자리
  - ▶ Ex) 포인터 배열의 요소가 100개라면 문자열은 최대 99개까지만 입력하고 마지막 배열 요소는 널 포인터

- 포인터 배열을 선언할 때

---

```
char *str[100] = {0};           // 포인터 배열을 널포인터로 초기화
```

---

- 저장된 문자열을 출력할 때

---

```
for(i = 0; str[i] != NULL; i++)    // 배열 요소가 널포인터가 아닌 동안 출력
{
    printf("%s\n", str[i]);
}
```

---

# MAIN 함수의 명령행 인수 사용

---

- ▶ 명령행 인수(command line argument)
  - ▶ 명령행에서 프로그램을 실행시킬 때 프로그램의 이름 외에도 프로그램에 필요한 정보를 함께 넘겨 주는 것
  - ▶ 운영체제가 명령행 인수를 프로그램의 main 함수로 넘기는 방법 통해 포인터로 동적 할당한 영역을 배열처럼 사용가능

## 예제 16-6 명령행 인수를 출력하는 프로그램

---

```
1. #include <stdio.h>
2.
3. int main(int argc, char **argv)    // 명령행 인수를 받을 매개변수
4. {
5.     int i;
6.
7.     for(i = 0; i < argc; i++)      // 인수 개수 만큼 반복
8.     {
9.         printf("%s\n", argv[i]);   // 인수로 받은 문자열 출력
10.    }
11.
12.    return 0;
13. }
```

실행 결과

```
C:\>mycommand first_arg second_arg
MYCOMMAND
first_arg
second_arg

C:\> _
```

# MAIN 함수의 명령행 인수 사용

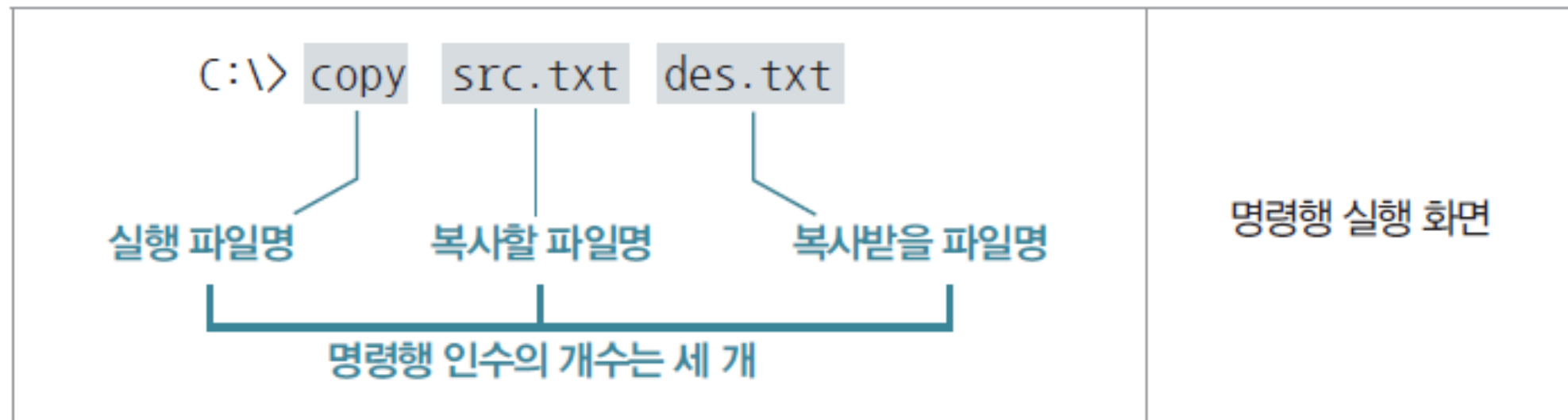
---

- ▶ main 함수는 명령행 인수 받기 위해 매개변수 선언
  - ▶ 매개변수의 이름은 임의 작성 가능
  - ▶ 관례적으로 argc와 argv 사용
  - ▶ 의미는 argument count, argument vector
  - ▶ 실행 결과와 같이 명령행 입력
    - ▶ 명령행 인수의 개수 3은 argc 매개변수에 저장
    - ▶ 명령행에서 입력한 문자열 위치는 argv 매개변수에 저장

# MAIN 함수의 명령행 인수 사용

---

- ▶ 운영체제가 명령행 문자열 처리하는 방법
  - ▶ 도스에서 사용하는 복사 프로그램
    - ▶ 복사할 파일과 복사 받을 파일의 이름 함께 입력
  - ▶ 이들이 모두 명령행 인수
- ▶ 명령행 인수의 수는 프로그램의 이름까지 포함하여 세 개



# MAIN 함수의 명령행 인수 사용

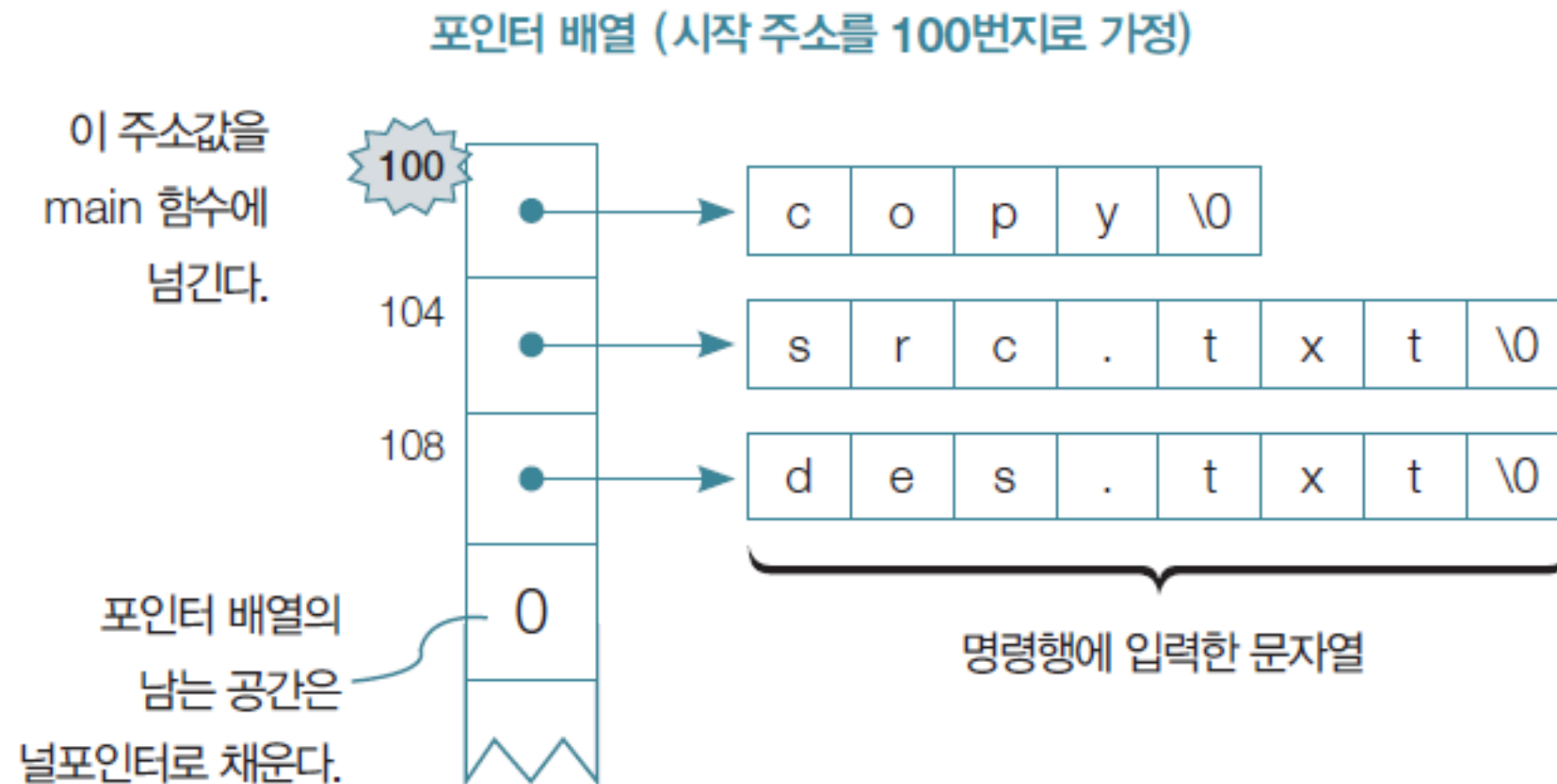
---

- ▶ 명령행에서 프로그램 실행
  - ▶ 운영체제는 명령행 인수를 가공하여 문자열의 형태로 메모리에 저장
  - ▶ 포인터 배열로 연결한 후에 포인터 배열의 시작 위치를 실행 프로그램의 main 함수에 넘김
  - ▶ 명령행 인수의 개수도 함께 전달

# MAIN 함수의 명령행 인수 사용

---

- ▶ 명령행에서 프로그램 실행될 때의 메모리 상황

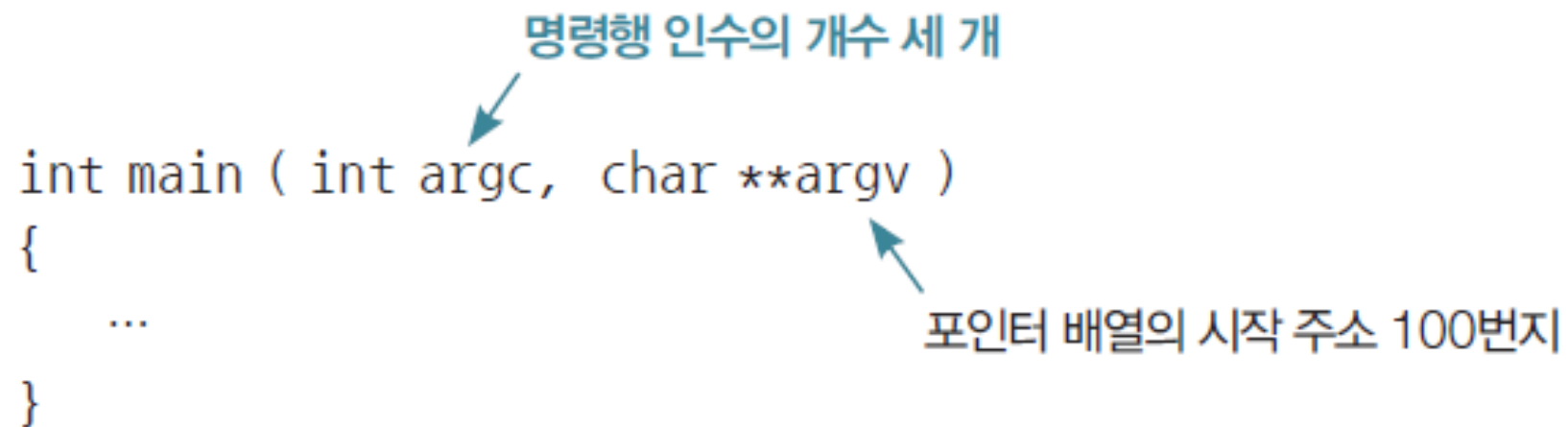




# MAIN 함수의 명령행 인수 사용

---

- ▶ 명령행에서 프로그램 실행될 때 인수 전달 상태



```
int main ( int argc, char **argv )  
{  
    ...  
}
```

The diagram illustrates the main function signature with two annotations:

- An arrow points from the text "명령행 인수의 개수 세 개" (Number of command-line arguments, three) to the `argc` parameter.
- An arrow points from the text "포인터 배열의 시작 주소 100번지" (Starting address of the pointer array, 100th) to the `argv` parameter.

# MAIN 함수의 명령행 인수 사용

---

- ▶ argc의 값이 아닌 널 포인터 활용하여 출력하는 방법
  - ▶ 명령행의 문자열을 연결하는 포인터 배열
    - ▶ 마지막 문자열 다음 배열 요소가 널 포인터
  - ▶ 널 포인터 아닌 동안 출력하도록 while문 작성 (argv의 값이 바뀌므로 명령행 인수 다시 사용하지 않도록 주의)

---

```
while(*argv != NULL)           // argv가 가리키는 배열 요소의 값이 널포인터가 아닌 동안
{
    printf("%s\n", *argv);      // 문자열을 출력
    argv++;                     // 포인터 배열의 다음 요소로 이동
}
```

---

---

```

1. #include <stdio.h>
2. #include <string.h>
3. #include <stdlib.h>           // malloc, atoi 함수 사용을 위한 헤더 파일           .....
4.
5. void print_str(char **);      // 문자열 출력 함수 선언
6.
7. int main(int argc, char **argv)
8. {
9.     char temp[80];             // 문자열 입력을 위한 임시 char 배열
10.    char **str;                // 포인터 배열로 사용할 동적 할당 영역 연결
11.    int max;                   // 최대 입력 문자열 수를 저장
12.    int i;
13.
14.    max = atoi(argv[1]);        // 두 번째 명령행 인수를 정수로 변환
15.    str = (char **) malloc((max + 1) * sizeof(char *)); // 포인터 배열의 동적 할당
16.    i = 0;
17.
18.    while(1)
19.    {
20.        printf("문자열을 입력하세요 : ");
21.        gets(temp);             // 문자열 입력
22.        if(temp[0] == '\0') break; // 엔터만 입력하면 반복 종료
23.
24.        str[i] = (char *) malloc(strlen(temp) + 1); // 문자열 저장 영역 동적 할당
25.        strcpy(str[i], temp);    // 문자열 복사
26.        i++;

```

```

27.         if(i == max)                // 입력된 문자열의 수를 검사
28.         {
29.             printf("문자열 입력이 최대값을 모두 채웠습니다.\n");
30.             break;
31.         }
32.     }
33.     str[i] = 0;                       // 입력이 끝난 후 널포인터로 마감
34.     print_str(str);                   // 입력된 문자열 출력
35.
36.     i = 0;
37.     while(str[i] != 0)
38.     {
39.         free(str[i]);                // 문자열을 저장한 동적 할당 영역 반환
40.         i++;
41.     }
42.     free(str);                        // 포인터 배열을 위해 동적 할당한 영역 반환
43.     return 0;
44. }
45.
46. void print_str(char **ps)            // 2중 포인터 ps는 포인터 배열처럼 사용
47. {
48.     while(*ps != 0)                  // ps 배열 요소의 값이 널포인터가 아닌 동안
49.     {
50.         printf("%s\n", *ps);         // ps 배열 요소가 연결하고 있는 문자열 출력
51.         ps++;                         // 다음 요소로 이동
52.     }
53. }

```

실행  
결과

[실.행.결.과 1]

C:\>strings 5

문자열을 입력하세요 : What's up!

문자열을 입력하세요 : Good morning~

문자열을 입력하세요 : Hi... \*^^\*

문자열을 입력하세요 :

What's up!

Good morning~

Hi... \*^^\*

[실.행.결.과 2]

C:\>strings 3

문자열을 입력하세요 : Cheer up.

문자열을 입력하세요 : Good evening.

문자열을 입력하세요 : Hello world!

문자열 입력이 최댓값을 모두 채웠습니다.

Cheer up.

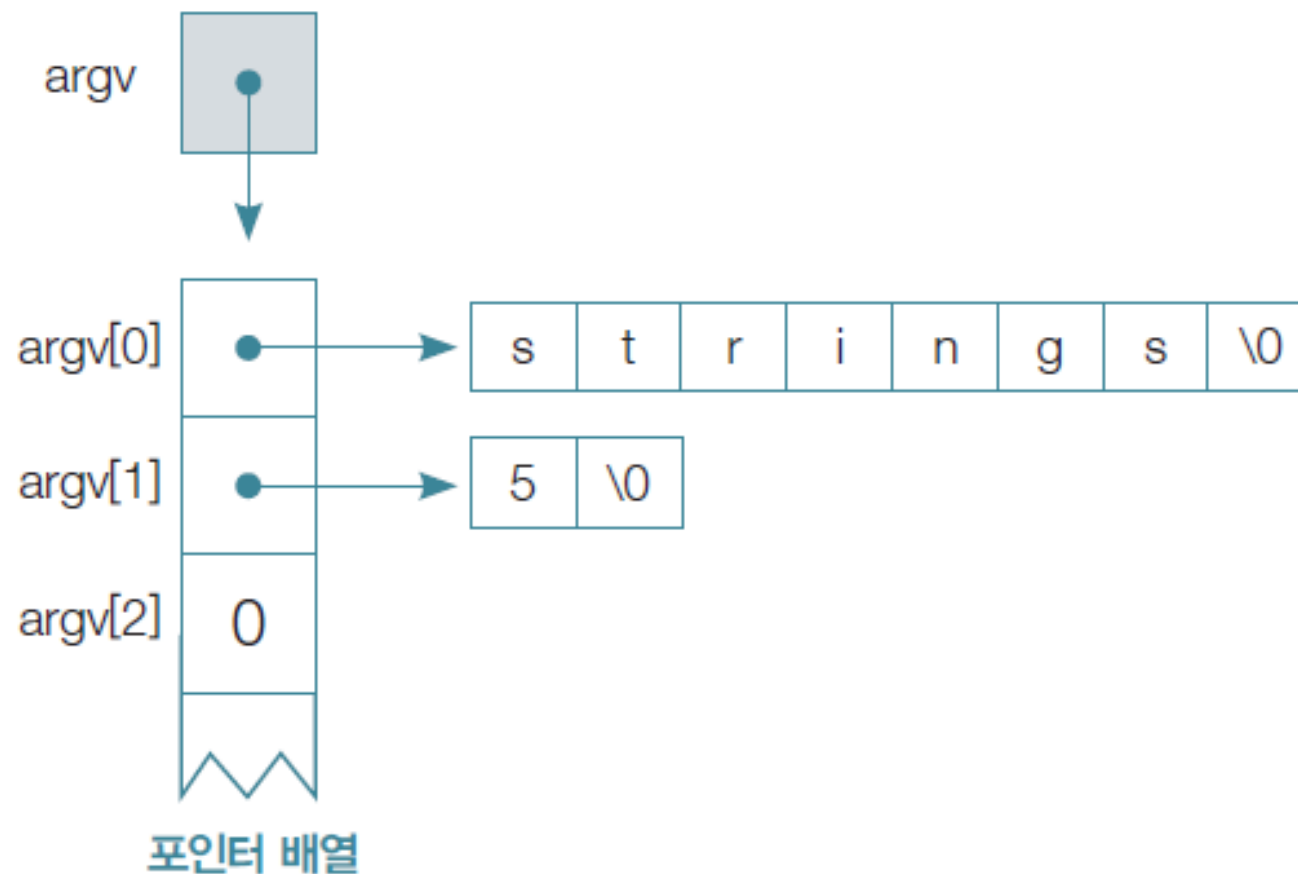
Good evening.

Hello world!

# 명령행 인수를 사용한 문자열 입력

---

- ▶ 명령행 인수를 통해 저장할 수 있는 문자열의 최대 수 결정
  - ▶ 실행할 때 명령행에서 두 번째로 입력하는 문자열은 `argv[1]`에 연결
  - ▶ 첫 번째 실행 예는 저장 공간이 메모리에 다음과 같이 연결



# 명령행 인수를 사용한 문자열 입력

---

- ▶ 프로그램은 argv[1]이 연결하고 있는 문자열 사용
  - ▶ 입력될 문자열을 연결할 포인터 배열의 크기로 사용
- ▶ argv[1]이 연결하고 있는 것은 문자열
  - ▶ 이 문자열을 정수로 바꾸는 과정 필요
- ▶ 숫자의 형태로 된 문자열 정수로 바꾸기 위함
  - ▶ atoi 함수 호출
    - ▶ 문자열의 주소를 인수로 받아서 문자열 정수로 바꾼 후에 반환하며 헤더 파일은 stdlib.h 사용

---

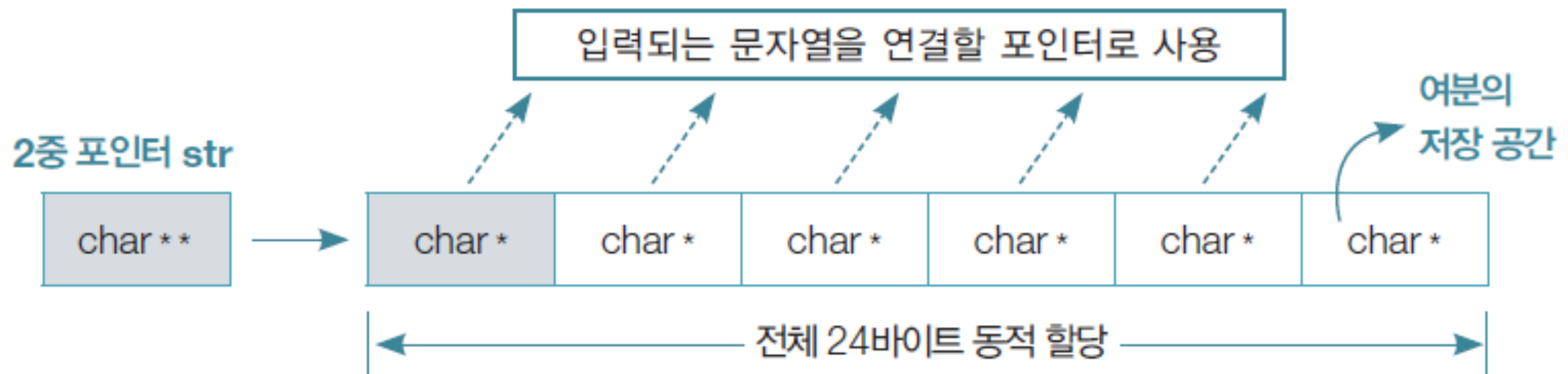
```
int atoi(char *);    // 문자열을 정수로 변환한 후 반환
```

---

# 명령행 인수를 사용한 문자열 입력

---

- ▶ 할당한 저장 공간을 2중 포인터 str로 연결
  - ▶ 포인터 배열처럼 사용할 저장 공간 만드는 과정 완료





# 명령행 인수를 사용한 문자열 입력

---

