CHAPTER

16

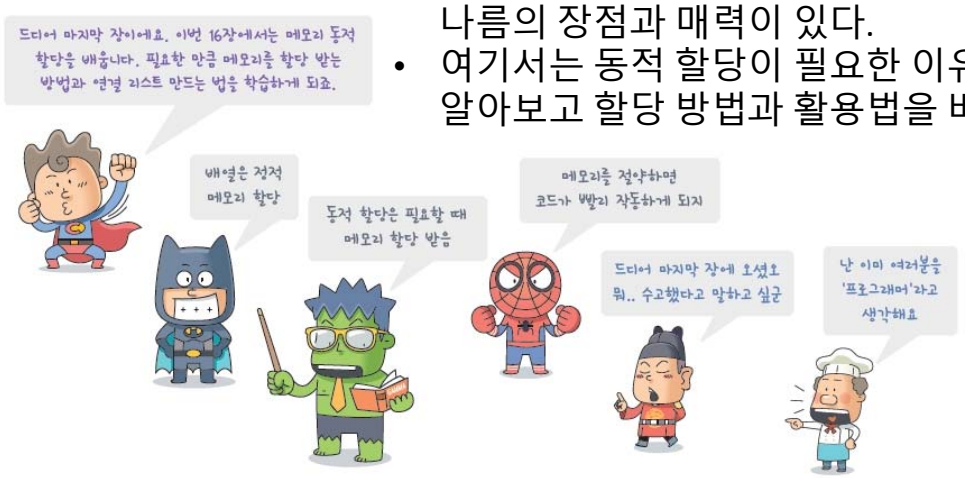
동적 메모리 할당

학습목차

- 16.1 동적으로 메모리 할당 받기
- 16.2 연결 리스트

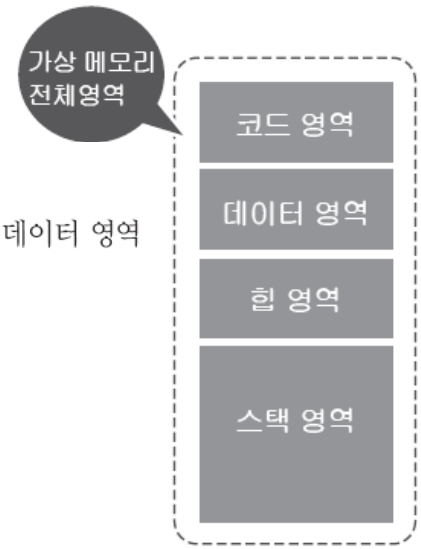
학습목표

- 메모리 동적 할당은 포인터를 사용해야 하고 메모리 누수를 신경 써야 하는 부담이 있다. 이는 변수나 배열과 같이 메모리를 할당하는 하나의 방법이며 나름의 장점과 매력이 있다.
- 여기서는 동적 할당이 필요한 이유를 알아보고 할당 방법과 활용법을 배워본다.



■ 메모리 구조

- 실행할 프로그램의 코드를 올려 놓을(저장할) 공간 : 코드 영역
- 프로그램이 종료될 때까지 유지해야 할 데이터를 저장할 공간 : 데이터 영역
- 아주 잠깐 사용하고 삭제할 데이터의 저장공간 : 스택 영역
- 프로그래머가 원하는 방식으로 쓸 수 있는 공간 : 힙 영역



동적 할당 메모리의 개념

- 프로그램이 메모리를 할당받는 방법
  - 정적(static)
  - 동적(dynamic)

정적 메모리 할당

- 정적 메모리 할당
  - 프로그램이 시작되기 전에 미리 정해진 크기의 메모리를 할당받는 것
  - 메모리의 크기는 프로그램이 시작하기 전에 결정
    - (예) int score_s[100];
  - 처음에 결정된 크기보다 더 큰 입력이 들어온다면 처리하지 못함
  - 더 작은 입력이 들어온다면 남은 메모리 공간은 낭비

동적 메모리 할당

- **동적 메모리 할당**
  - 실행 도중에 동적으로 메모리를 할당받는 것
  - 사용이 끝나면 시스템에 메모리를 반납
  - score = (int *)malloc(100*sizeof(int));
  - 필요한 만큼만 할당을 받고 메모리를 효율적으로 사용

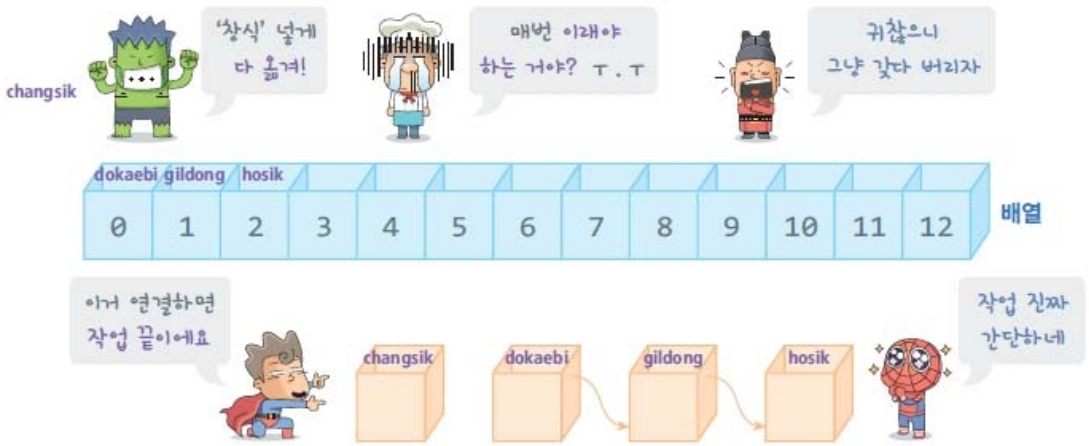
16.1 동적으로 메모리 할당 받기

1 동적 메모리 할당의 필요성

- 배열은 한번 선언되면 크기와 시작 주소를 변경 할 수 없음
  - 정적 메모리 할당(static memory allocation) 방식
- 메모리를 필요한 만큼 제공 받고 필요가 없을 때 메모리를 반환하는 방법
  - 동적 메모리 할당(dynamic memory allocation) 방식

16.1 동적으로 메모리 할당 받기

- 배열에 저장되어 있던 사람이름을 삭제하는 경우, 배열에서는 삭제된 데이터가 있던 자리가 빈 공간으로 남게 됨
- 동적 할당에서는 필요 없는 메모리를 없앨 수 있음 - 낭비되는 공간이 줄어들게 됨
- 이름순으로 저장되어 있는 데이터에 새로운 이름이 들어오는 경우, 배열에서는 순서에 맞도록 데이터를 이동 시켜야 함 - 데이터가 많을 경우 이러한 작업은 매우 불편
- 동적 메모리 할당에서는 새로운 데이터를 만든 후 연결만 시키면 됨
- 데이터의 삽입이나 삭제의 경우, 동적 메모리 할당에서는 데이터의 이동 없이 순서대로 이름을 관리 할 수 있음



16.1 동적으로 메모리 할당 받기

2 동적 할당 함수

- 동적 메모리 할당의 핵심이 되는 함수는 malloc()
- 메모리를 할당 받을 때에도 정수로 사용할 메모리 공간인지 실수로 사용할 메모리 공간인지를 알려주어야 함
- 동적으로 만들어진 메모리 공간을 접근할 때는 포인터를 사용
- 정수 포인터 *dmp를 먼저 선언 하고 난 후, malloc() 함수를 사용

동적 메모리 할당 malloc()

```
#include <stdlib.h>
int *dmp;
dmp = (int *)malloc(sizeof(int));
```

- 일반적인 포인터 사용과 동일

```
*dmp = 100;
printf("%d\n", *dmp);
```

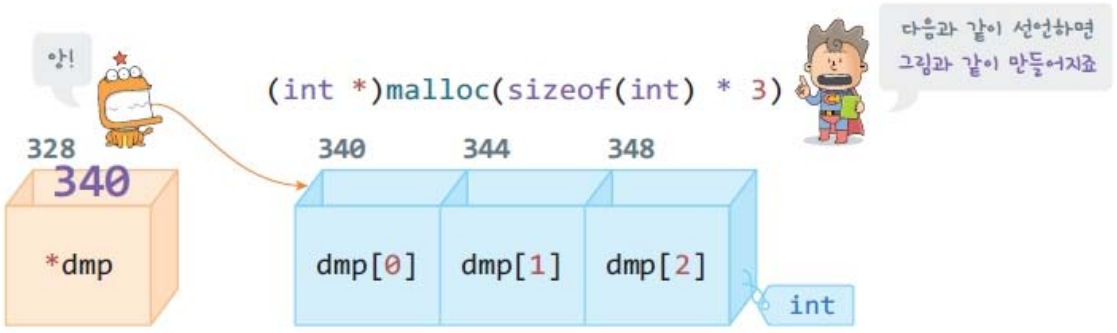
동적 메모리 할당 이후에는
일반적인 포인터 사용 방법과 같다.

16.1 동적으로 메모리 할당 받기

- 만약 배열을 선언하고자 하는 경우, malloc()을 사용할 때 할당되는 메모리의 크기만 늘려주면 됨

동적 메모리 할당에서 배열 선언

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



16.1 동적으로 메모리 할당 받기

- 동적 메모리 할당 받은 후 사용 방법

```
dmp[0] = 100;
dmp[1] = 200;
dmp[3] = 300;
```

```
printf("%d\n", *dmp);
printf("%d\n", *(dmp + 1));
printf("%d\n", *(dmp + 2));
```

- 동적 메모리 반환 – free()

동적 메모리 반납 free()

```
free(dmp);
```

16.1 동적으로 메모리 할당 받기

[예제 16-1] 동적 메모리 할당 코드

```c
#include <stdio.h>
#include <stdlib.h>                    // 동적 메모리 함수가 들어 있는 헤더

int main() {
   int max, k;
   double *dmp, sum = 0.0;

   printf("입력하는 숫자는 몇 개? ");
   scanf("%d", &max);
   dmp = (double *)malloc(sizeof(double) * max);      // 동적 메모리 할당

   printf("%d개 값을 입력하시오 :", max);
   for (k = 0; k < max; k++) {             // k는 0부터 max보다 작을 때까지
      scanf("%lf", &dmp[k]);
      sum = sum + dmp[k];
   }
   free(dmp);                              // 동적 할당 메모리 반환
   printf("평균 = %.1lf", sum / max);

   return 0;
}
```

실행 화면

```
입력하는 숫자는 몇 개? 3
3개 값을 입력하시오 :22 56 24
평균 = 34.0
```


[malloc 함수와 free 함수의 매크로화]

- 아래의 코드는 malloc 함수의 일반적인 사용 예이다.

```
int * ptr;      #define MALLOC(x, y) (x *)malloc(sizeof(x)*(y));
ptr = (int *)malloc(sizeof(int)*5);
```

여기서 malloc 함수의 호출 과정을 다음과 같이 간단히 할 수 있도록 MALLOC이라는 이름의 매크로 함수를 정의하자. 즉 형 변환과 sizeof 연산, 그리고 곱셈 연산의 과정을 생략할 수 있도록 매크로 함수를 정의하면 된다.

```
int * ptr;
ptr = MALLOC(int, 5);
```

아래의 코드가 하는 역할을 한줄의 코드가 대신할 수 있도록 매크로 함수를 정의하라.

```
free(ptr); // FREE(ptr);      #define FREE(x) free(x); x = NULL;
ptr = NULL;
```

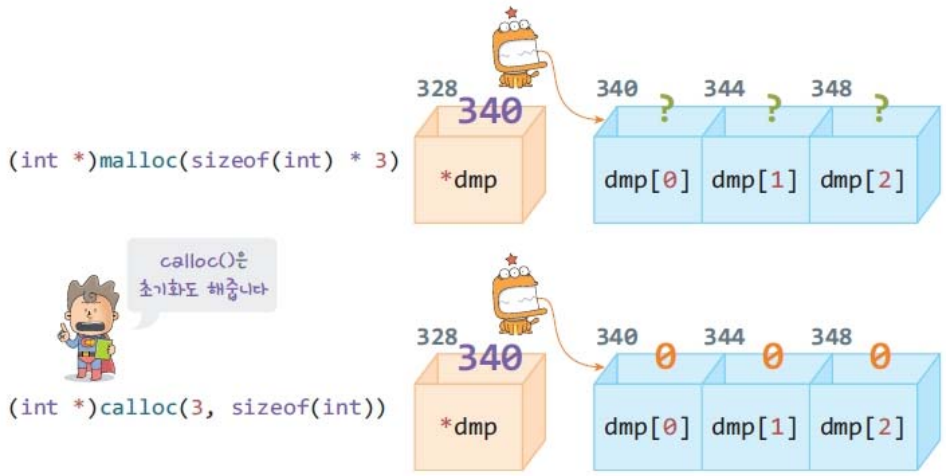
3 calloc()과 realloc()

- 0으로 초기화된 동적 메모리 할당 함수가 calloc()
- calloc()은 malloc()과 거의 유사하지만, 메모리 전체 갯수를 먼저 기술하고, 메모리 한 개의 크기를 다음에 기술

초기화된 동적 메모리 할당 calloc()

```
dmp = (int *)calloc(size, sizeof(int));
```

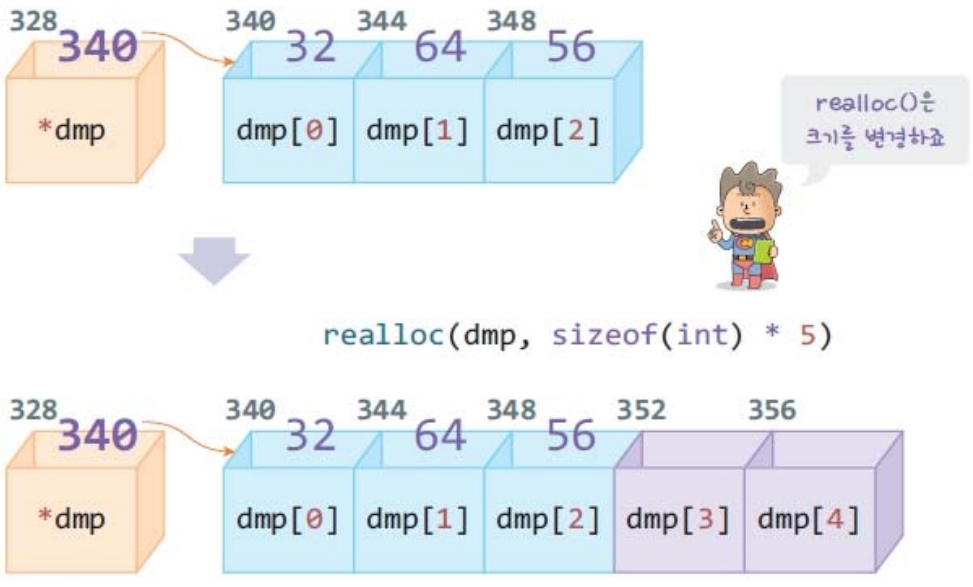
- calloc()과 malloc()의 차이



- realloc()은 할당 받은 메모리의 크기를 변경 시켜줌
- 동적 메모리의 크기를 변화 시킬 뿐이기 때문에 기존의 자료형은 그대로 유지

메모리 크기 변경 realloc()

```
dmp = realloc(dmp, sizeof(int) * size);
```



- calloc 함수는 할당한 공간을 0으로 초기화 한다.

calloc 함수 원형

개수 × 단위크기

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

반환값 — 매개변수

- realloc 함수는 할당한 공간의 크기를 늘이거나 줄인다.

realloc 함수 원형

대상 변경할 크기

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

반환값 — 매개변수

16.1 동적으로 메모리 할당 받기

[예제 16-2] 동적 메모리 할당 코드

```c
#include <stdio.h>
#include <stdlib.h>                    // 동적 메모리 함수가 들어 있는 헤더

int main() {
   int max, add, k;
   double *dmp, sum = 0.0;

   printf("입력하는 숫자는 몇 개? ");
   scanf("%d", &max);
   dmp = (double *)calloc(max, sizeof(double));// calloc()으로 동적 할당

   printf("%d개 값을 입력하시오 :", max);
   for (k = 0; k < max; k++) {             // k는 0부터 max보다 작을 때까지
      scanf("%lf", &dmp[k]);
      sum = sum + dmp[k];
   }
   printf("추가 숫자는 몇 개? ");
   scanf("%d", &add);
   dmp = realloc(dmp, sizeof(double) * (max + add)); // 메모리 크기 변경

   printf("추가되는 %d개 값을 입력하시오 :", add);
   for (k = max; k < (max + add); k++) {// max부터 max+add보다 작을 때까지
      scanf("%lf", &dmp[k]);
      sum = sum + dmp[k];
   }
   free(dmp);                                    // 동적 할당 메모리 반환
   printf("평균 = %.1lf", sum / (max + add));

   return 0;
}
```

실행 화면

```
입력하는 숫자는 몇 개? 3
3개 값을 입력하시오 :11 22 33
추가 숫자는 몇 개? 2
추가되는 2개 값을 입력하시오 :44 55
평균 = 33.0
```

마무리

❖ 다음 용도에 맞게 저장 공간을 동적 할당하여 각 포인터에 연결하라.

① 몸무게를 저장할 변수

```
double *weight =  (double *)malloc(sizeof(double));
```

② 열 과목의 점수를 저장할 배열

```
int *scores =   (int *)calloc(10, sizeof(int));
```

③ 널 문자를 포함하여 최대 80글자의 문자열을 저장할 char 배열

```
char *string =   (char *)malloc(80 * sizeof(char));
```


마무리

❖ 동적 할당이 제대로 되었는지 검사하고 사용이 끝난 동적 할당 영역을 반환하도록 빈칸을 채워라.

```
int *max = (int *)malloc(sizeof(int));
if (        ①        )   max == NULL
{
        printf("메모리가 부족합니다.");
        return 1;
}
*max = 999;
(        ②        )   free(max);
```

마무리

❖ 다음 코드에서 pa, pb, pc에 할당되는 동적 할당 영역의 크기를 쓰시오.

```
int ary[5] = {1, 2, 3, 4, 5};
int *pa, *pb, *pc;
int i;
pa = (int *)malloc(sizeof(ary));
for (i=0; i < 5; i++) {
        pa[i] = ary[i];
}
pb = (int *)calloc(pa[3], sizeof(int));
pc = (int *)realloc(NULL, pa[4]);
```


01 다음 중 동적 메모리 할당 함수 중 초기화되지 않은 상태로 메모리를 할당받는 함수는 무엇인가?

① `malloc()` ② `realloc()` ③ `calloc()` ④ `free()`

02 다음 중 동적 메모리 할당 함수를 사용하기 위해서는 어떤 헤더를 포함해야 하는가?

① `string.h` ② `stdio.h` ③ `time.h` ④ `stdlib.h`

03 다음 중 동적 메모리 할당 함수 중 초기화된 상태로 메모리를 할당받는 함수는 무엇인가?

① `malloc()` ② `realloc()` ③ `calloc()` ④ `free()`

04 다음 중 동적 메모리 할당 함수 중 할당된 메모리를 반환할 때 사용하는 함수는 무엇인가?

① `malloc()` ② `realloc()` ③ `calloc()` ④ `free()`

05 다음 중 동적 메모리 할당 함수 중 할당된 메모리의 크기를 변경할 때 사용하는 함수는 무엇인가?

① malloc()　② realloc()　③ calloc()　④ free()

06 다음 중 정수 한 개를 동적 메모리 할당으로 받으려 할 때 옳게 선언한 코드는 무엇인가?

① (int)malloc(sizeof(int))　② (int *)malloc(sizeof(int))

③ malloc(sizeof(int))　④ (int *)malloc(sizeof int)

07 다음 중 문자 한 개를 동적 메모리 할당으로 받으려 할 때 옳게 선언한 코드는 무엇인가?

① (char *)malloc(sizeof(int))　② (int *)malloc(sizeof(char))

③ (char *)malloc(sizeof(char))　④ (char)malloc(sizeof(char))

08 다음 중 실수 한 개를 동적 메모리 할당으로 받으려 할 때 옳게 선언한 코드는 무엇인가?

① (float *)malloc(sizeof(float))　② (float *)malloc(sizeof(char))

③ (int *)malloc(sizeof(char))　④ (float)malloc(sizeof(char))

09 다음 중 정수 3개 배열을 동적 메모리 할당으로 받으려 할 때 옳게 선언한 코드는 무엇인가?

① (int* 3)malloc(sizeof(int))　② (int* 3)malloc(sizeof(int) * 3)

③ (int)malloc(sizeof(int) * 3)　④ (int *)malloc(sizeof(int) * 3)

10 다음 중 calloc()을 사용하여 정수 3개 배열을 동적 메모리 할당으로 받으려 할 때 옳게 선언한 코드는 무엇인가?

① (int * 3)calloc(sizeof(int))　② (int * 3)calloc(3, sizeof(int))

③ (int *)calloc(3, sizeof(int) * 3)　④(int*)calloc(3, sizeof(int))

11 다음 중 동적으로 할당 받은 메모리 dmp의 크기를 5로 늘리려 한다. realloc()을 옳게 선언한 코드는 무엇인가?

① dmp = realloc(dmp, sizeof(int) * 3)

② dmp = realloc(dmp, sizeof(int) * 5)

③ dmp = realloc(5, sizeof(int))

④ dmp = realloc(5, sizeof(int) * 3)

12 다음 중 malloc() 함수의 반환(return) 값은 무엇으로 선언되어 있는가?

① void ② 포인터 ③ void 포인터 ④ 이중 포인터

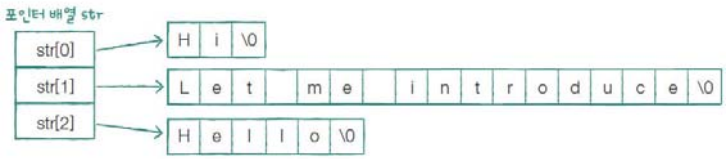
13 다음 중 calloc() 함수의 반환(return) 값은 무엇으로 선언되어 있는가?

① void ② 포인터 ③ void 포인터 ④ 이중 포인터

- **동적 할당을 사용한 문자열 처리**

3개의 문자열을 저장하기 위한 동적 할당 소스 코드 예제16-4.c

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 #include <string.h>
04
05 int main(void)
06 {
07     char temp[80];
08     char *str[3];
09     int i;
10
11     for (i = 0; i < 3; i++)
12     {
13         printf("문자열을 입력하세요 : ");
14         gets(temp);
15         str[i] = (char *)malloc(strlen(temp) + 1);
16         strcpy(str[i], temp);      // 동적 할당 영역0
17     }
18
19     for (i = 0; i < 3; i++)
20     {
21         printf("%s\n", str[i]);
22     }
23
24     for (i = 0; i < 3; i++)
25     {
26         free(str[i]); // 동적 할
27     }
28
29     return 0;
30 }
```



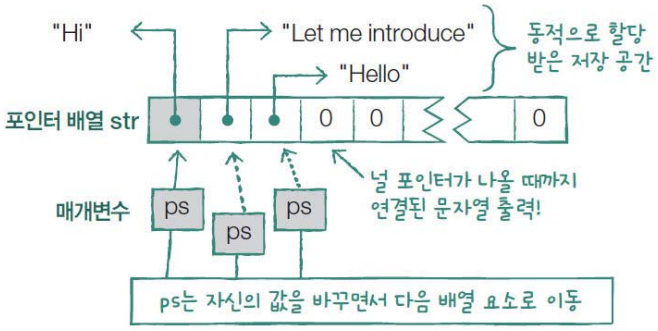
실행결과

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

- **동적 할당 영역에 저장한 문자열을 함수로 처리하는 예**

동적 할당 영역의 문자열을 함수로 출력 소스 코드 예제16-5.c

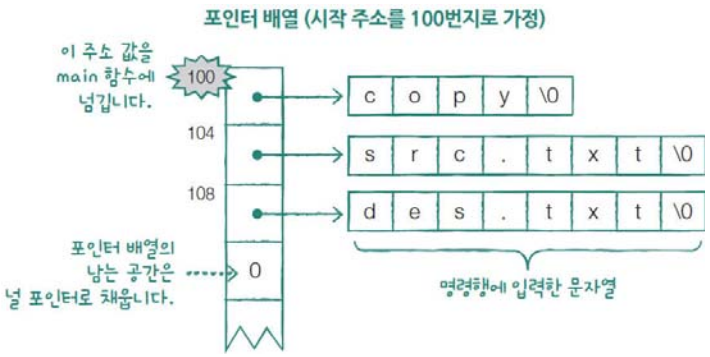
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void print_str(char **ps);

int main(void)
{
    char temp[80];
    char *str[21] = { 0 };
    int i = 0;

    while (i < 20)
    {
        printf("문자열을 입력하세요 : ");
        gets(temp);
        if (strcmp(temp, "end") == 0) break;
        str[i] = (char *)malloc(strlen(temp) + 1);
        strcpy(str[i], temp);
        i++;
    }
    print_str(str);

    for (i = 0; str[i] != NULL; i++)
    {
        free(str[i]);
    }

    return 0;
}

void print_str(char **ps)
{
    while (*ps != NULL)
    {
        printf("%s\n", *ps);
        ps++;
    }
}
```



- **main 함수의 명령행 인수 사용**

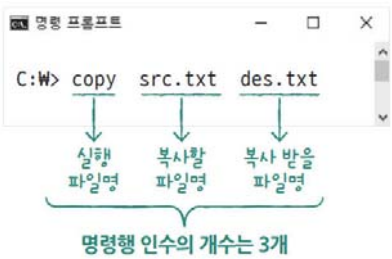
명령행 인수를 출력하는 프로그램 소스 코드 예제16-6.c

```c
#include <stdio.h>

int main(int argc, char **argv)
{
    int i;

    for (i = 0; i < argc; i++)
    {
        printf("%s\n", argv[i]);
    }

    return 0;
}
```



실행결과

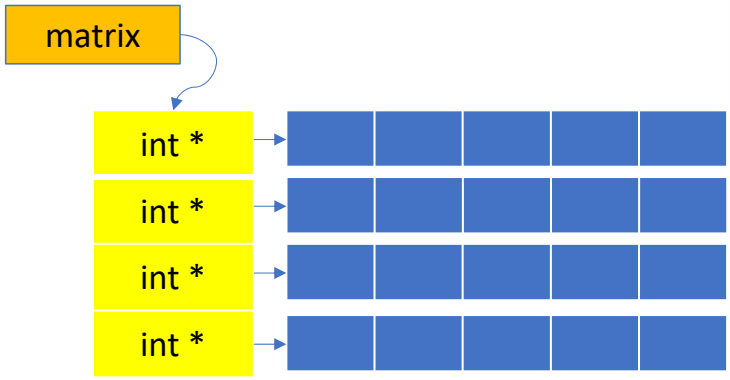
```
C:W> cd C:WstudyCW16-6WDebug
C:WstudyCW16-6WDebug> 16-6 first_arg second_arg
16-6
first_arg
second_arg
C:WstudyCW16-6WDebug> exit
```



마무리

❖ 4행 5열의 행렬의 값을 저장할 2차원 배열을 동적 할당하고 해제하는 코드를 이용하는 프로그램을 작성하라.

```
int **matrix = (int **) malloc(4 * sizeof(int *));
for (i=0; i < 4; i++) {
        matrix[i] = (int *) malloc(5 * sizeof(int));
}
// 사용 프로그램 작성
for (i = 0; i < 4; i++) {
        free(matrix[i]);
}
free(matrix);
```



마무리

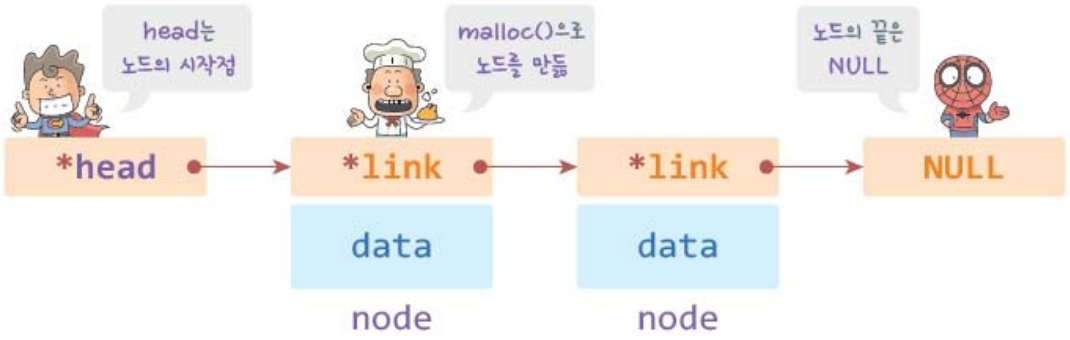
❖ 다음은 명령행 인수를 사용하여 프로그램 이름을 제외한 나머지 인수를 출력하는 코드이다. 빈칸을 완성하라.

```
int main(          (1)          )    int argc, char ** argv
{
        int i;
        for (i=0; i <   (2)      ; i++) {  argc - 1
                printf("%s\n",    (3)      );   *(argv + i + 1), argv[i+1]
        }
}
```

16.2 연결 리스트

1 연결 리스트에 대하여

- 연결리스트에서 하나의 구조체는 노드(node)라 부름
- 노드는 데이터를 담은 데이터 영역과 다른 노드를 가리키는 링크(link) 영역으로 나뉨
- 연결 리스트의 구조는 다음과 같음
- 보통의 경우 연결 리스트의 시작을 *head가 가지고 있음
- *head는 연결 리스트의 처음 노드를 가리키는 포인터
- 연결 리스트의 맨 끝은 항상 NULL



16.2 연결 리스트

- 그림은 노드의 정의 예
- typedef를 사용하여 struct node를 정의
- node 구조체에는 데이터의 영역과 link 영역으로 나뉨
- 문자열 name과 정수 age가 데이터이고, node 구조체를 가리키는 포인터로 *link가 정의됨

연결 리스트 노드 정의

```
typedef struct node { // typedef로 node 구조체 정의
char name[10];        // node의 data 1
int age;              // node의 data 2
struct node *link;    // 자기 자신을 가리키는 *link 정의
} node;
```

16.2 연결 리스트

- 구조체 포인터 *p1을 정의하고, malloc()을 사용하여 노드를 생성
- 동적으로 할당 받은 메모리 주소를 p1에게 넘겨 주면 됨

```
node *p1;
p1 = (node *)malloc(sizeof(node));
```

- 노드 안에 있는 데이터는 구조체의 값을 대입하는 방법

```
strcpy(p1->name, "gildong");
p1->age = 22;
p1->link = NULL;
```


16.2 연결 리스트

2 연결 리스트의 구현

[예제 16-3] 연결 리스트 코드

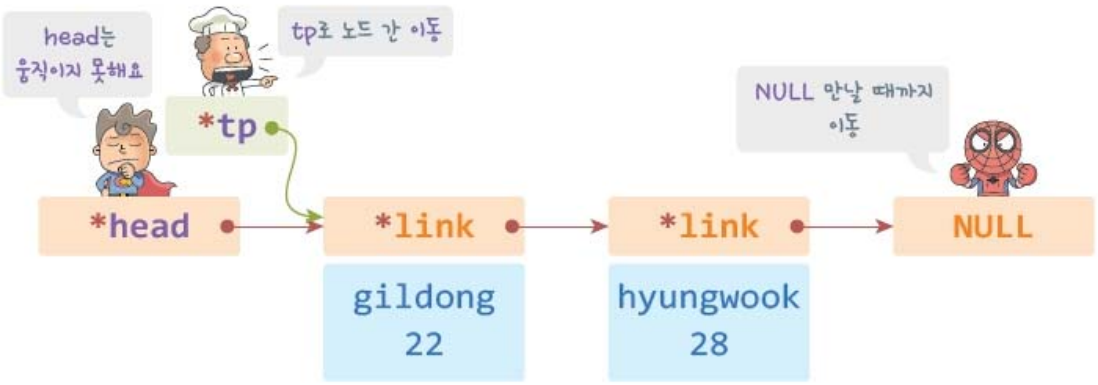
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct node {                  // typedef로 node 구조체 정의
    char name[10];                     // node의 data 1
    int age;                           // node의 data 2
    struct node *link;
 } node;

int main() {
   node *head = NULL, *np, *tp;

   np = (node *)malloc(sizeof(node));
   strcpy(np->name, "gildong");
   np->age = 22;
   np->link = NULL;
   head = np;                          //
```

```

   np = (node *)malloc(sizeof(node));
   strcpy(np->name, "hyungwook");
   np->age = 28;
   np->link = NULL;
   head->link = np;                    // 새로운 노드를 연결 리스트 맨 뒤에 연결

   tp = head;
   while (tp != NULL) {
      printf("[%s:%d] -> ", tp->name, tp->age);
      tp = tp->link;                   // tp를 연결 리스트 다음 노드로 이동
   }
   printf("NULL\n");
   return 0;
}
```

실행 화면

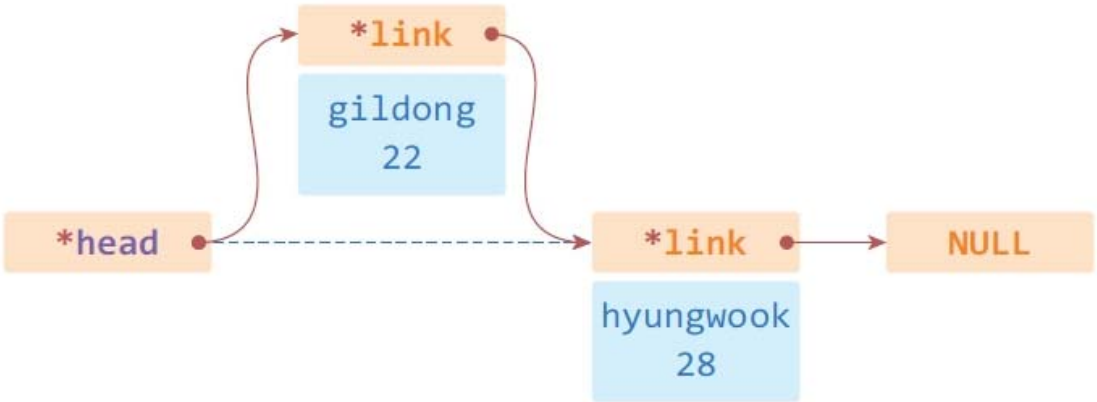
```
[gildong:22] -> [hyungwook:28] -> NULL
```

- 20번 줄에서 23번 줄까지는 두 번째 node를 만들어 값을 대입하는 코드
- 20번 줄에서 새로운 노드를 생성
- 21번 줄에서 "hyungwook"을 name에 넣음
- 22번 줄에서 age에 28을 넣음
- 23번 줄에서 node->link를 NULL로 초기화 함
- 24번 줄에서 head->link를 np와 연결
- head->link는 처음에 만든 gildong node의 link임 - 그림과 같은 모양이 됨



3 연결 리스트의 삽입

- 사용자로부터 값을 입력 받아 연결 리스트를 만들고 출력하는 코드 만들기



16.2 연결 리스트

[예제 16-4] 역순 연결 리스트 삽입 코드 : 예제 16-5 순방향 연결과의 비교 분석

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct node {                  // typedef로 node 구조체 정의
    char name[10];                     // node의 data 1
    int age;                           // node의 data 2
    struct node *link;                 // 자기 자신을 가리키는 *link 정의
 } node;

int main() {
   node *head = NULL, *np, *tp;
   int k;

   for (k = 0; k < 3; k++) {
      np = (node *)malloc(sizeof(node));
      printf("이름 나이 입력: ");
      scanf("%s %d", &np->name, &np->age);
      np->link = NULL;

```


16.2 연결 리스트

[예제 16-4] 역순 연결 리스트 삽입 코드 (계속)

```c
      if (head == NULL)                          // head가 처음 연결이면
         head = np;
      else { np->link = head;                    // head에 노드가 있으면
             head = np; }
   }                                             // End of for
   tp = head;
   while (tp != NULL) {
      printf("[%s:%d] -> ", tp->name, tp->age);
      tp = tp->link;                         // tp를 연결 리스트 다음 노드로 이동
   }
   printf("NULL\n");
   return 0;
}
```

실행 화면

```
이름 나이 입력: gildong 22
이름 나이 입력: hyungwook 28
이름 나이 입력: hosik 29
[hosik:29] -> [hyungwook:28] -> [gildong:22] -> NULL
```

16.2 연결 리스트

[예제 16-5] 순방향 연결 리스트 삽입 코드

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct node {              // typedef로 node 구조체 정의 해야 함
    char name[10];                 // node의 data 1
    int age;                       // node의 data 2
    struct node *link;             // 자기 자신을 가리키는 *link 정의
 } node;

int main() {
   node *head = NULL, *np, *tp;
   int k;

   for (k = 0; k < 3; k++) {
      np = (node *)malloc(sizeof(node));
      printf("이름 나이 입력: ");
      scanf("%s %d", &np->name, &np->age);
      np->link = NULL;
```


16.2 연결 리스트

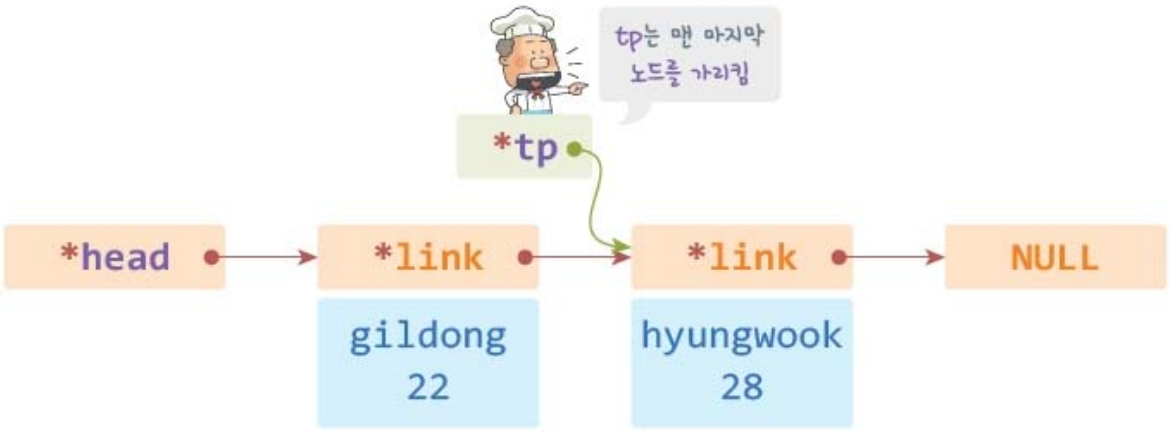
[예제 16-5] 순방향 연결 리스트 삽입 코드 (계속)

```c
      if (head == NULL) {
         head = np;
         tp = head; }
      else { tp->link = np;                    // head에 노드가 있으면
            tp = tp->link; }
   }                                        // End of for
   tp = head;
   while (tp != NULL) {
      printf("[%s:%d] -> ", tp->name, tp->age);
      tp = tp->link;
   }
   printf("NULL\n");
   return 0;
}
```

실행 화면

```
이름 나이 입력: gildong 22
이름 나이 입력: hyungwook 28
이름 나이 입력: hosik 29
[gildong:22] -> [hyungwook:28] -> [hosik:29] -> NULL
```

- hyungwook 노드 삽입 후 tp의 위치



14 다음 중 연결 리스트의 구조체에서 맨 앞을 가리키고 있는 것은 무엇인가?

① *head ② NULL

③ node의 데이터 영역 ④ struct

15 다음 중 연결 리스트의 구조체에서 맨 끝을 나타내는 것은 무엇인가?

① *head ② NULL

③ node의 데이터 영역 ④ struct

[객관식]

16 동적 메모리 할당 함수 중 초기화되지 않은 상태로 메모리를 할당받는 함수는 ( )이다.

17 동적 메모리 할당 함수를 사용하기 위해 포함해야 하는 헤더는 ( )이다.

18 동적 메모리 할당 함수 중 초기화된 상태로 메모리를 할당받는 함수는 ( ) 이다.

19 동적 메모리 할당 함수 중 할당된 메모리를 반환할 때 사용하는 함수는 ( ) 이다.

20 동적 메모리 할당 함수 중 할당된 메모리의 크기를 변경할 때 사용하는 함수는 ( )이다.

21 `malloc()`을 사용하여 정수 한 개를 동적 메모리 할당으로 받으려 할 때는 ( )와 같이 선언해야 한다.

22 `malloc()`을 사용하여 문자 한 개를 동적 메모리 할당으로 받으려 할 때는 ( )와 같이 선언해야 한다.

23 `malloc()`을 사용하여 실수 한 개를 동적 메모리 할당으로 받으려 할 때는 ( )와 같이 선언해야 한다.

24 `malloc()`을 사용하여 정수 3개 배열을 동적 메모리 할당으로 받으려 할 때 ( )과 같이 선언해야 한다.

25 `calloc()`을 사용하여 정수 3개 배열을 동적 메모리 할당으로 받으려 할 때는 ( )와 같이 선언해야 한다.

26 동적으로 할당 받은 메모리 `dmp`의 크기를 `5`로 늘리려 `realloc()`을 사용하는 경우 `dmp = (  )`와 같이 선언해야 한다.

27 연결 리스트의 구조체에서 맨 앞을 가리키고 있는 것은 ( )이다.

28 `malloc()` 함수의 반환(return) 값은 ( )로 선언되어 있다.

29 `calloc()` 함수의 반환(return) 값은 ( )로 선언되어 있다.

마무리

소수(prime number) 계산 프로그램

❖ 키보드로 양수를 입력한 후에 입력한 수까지의 모든 소수를 출력한다. 2부터 한 줄에 5개씩 출력하며 소수가 아닌 수는 X를 출력한다. 입력한 수에 따라 적절한 크기의 배열을 동적 할당하여 사용한다.

//실행 예
양수 입력 : 12

2 3 X 5 X
7 X X X 11

[malloc과 free의 활용]

- 최대 길이가 20을 넘지 않는 3개의 문자열을 저장할 수 있는 2차원 배열을 힙에 할당하고, 프로그램 사용자로부터 3개의 문자열을 이 배열에 입력받아서, 입력된 순서대로 출력하는 프로그램을 작성하라.