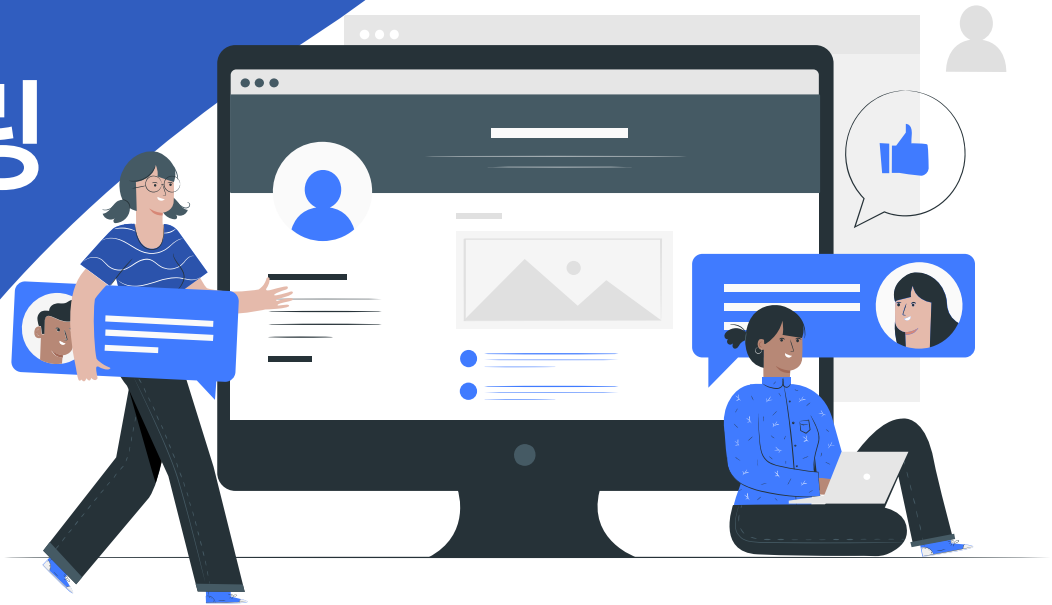
7주차 튜터링



연결리스트
실습 코드
그림으로 설명

목차

- 전체 코드
- 그림으로 설명

전체 코드

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

// 연결 리스트의 노드를 나타내는 구조체
typedef struct node
{
    int data;
    struct node* next;
} NODE;

// 새로운 노드를 생성하는 함수
NODE* create_node(int value)
{
    NODE* new_node = (NODE*)malloc(sizeof(NODE));

    if (new_node == NULL)
    {
        printf("메모리 할당 오류\n");
        exit(EXIT_FAILURE);
    }
    new_node->data = value;
    new_node->next = NULL;

    return new_node;
}

// 연결 리스트에 노드를 추가하는 함수
void insert_node(NODE** head, int value)
{
    NODE* new_node = create_node(value);

    if (*head == NULL)
    {
        // 리스트가 비어있을 경우
        *head = new_node;
    }
    else
    {
        // 리스트가 비어있지 않을 경우
        NODE* temp = *head;
        while (temp->next != NULL)
        {
            temp = temp->next;
        }
        temp->next = new_node;
    }
}

// 연결 리스트의 노드를 출력하는 함수
void print_list(NODE* head)
{
    NODE* temp = head;
    while (temp != NULL)
    {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

// 메모리 해제 함수
void free_list(NODE* head)
{
    NODE* temp;
    while (head != NULL)
    {
        temp = head;
        head = head->next;
        free(temp);
    }
    printf("메모리 해제 완료\n");
}

int main()
{
    NODE* my_list = NULL;

    // 연결 리스트에 노드 추가
    insert_node(&my_list, 10);
    insert_node(&my_list, 20);
    insert_node(&my_list, 30);
    insert_node(&my_list, 40);

    // 연결 리스트 출력
    printf("연결 리스트: ");
    print_list(my_list);

    // 메모리 해제
    free_list(my_list);

    return 0;
}
```

1. 그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  while(head != NULL)
    {
      temp = head;
      head = head->next;
      free(temp);
    }
  printf("메모리 해제 완료₩n");
}

int main()
{
  NODE *my_list = NULL;

  // 연결 리스트에 노드 추가
  insert_node(&my_list, 10);
  insert_node(&my_list, 20);
  insert_node(&my_list, 30);

  // 연결 리스트 출력
  printf("연결 리스트: ");
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

Stack | Heap

main
my_list | pointer to NODE ?

Note: ? refers to an uninitialized value

C/C++ details: none [default view]

2 그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  while(head != NULL)
    {
      temp = head;
      head = head->next;
      free(temp);
    }
  printf("메모리 해제 완료₩n");
}

int main()
{
  NODE *my_list = NULL;

  // 연결 리스트에 노드 추가
  insert_node(&my_list, 10);
  insert_node(&my_list, 20);
  insert_node(&my_list, 30);

  // 연결 리스트 출력
  printf("연결 리스트: ");
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

Stack | Heap

main
my_list | pointer to NODE ?

Note: ? refers to an uninitialized value

C/C++ details: none [default view]

3

그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  while(head != NULL)
    {
      temp = head;
      head = head->next;
      free(temp);
    }
  printf("메모리 해제 완료₩n");
}

int main()
{
  NODE *my_list = NULL;

  // 연결 리스트에 노드 추가
  insert_node(&my_list, 10);
  insert_node(&my_list, 20);
  insert_node(&my_list, 30);

  // 연결 리스트 출력
  printf("연결 리스트: ");
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

Stack Heap

main

my_list | pointer to NODE NULL (0x0)

C/C++ details: none [default view]

4

그림으로 설명

C (C17 + GNU extensions)
known limitations

```
20        exit(EXIT_FAILURE);
21      }
22    new_node->data = value;
23    new_node->next = NULL;
24
25    return new_node;
26  }
27
28  // 연결 리스트에 노드를 추가하는 함수
29  void insert_node(NODE **head, int value)
30  {
31    NODE *new_node = create_node(value);
32
33    if(*head == NULL)
34      {
35        // 리스트가 비어있을 경우
36        *head = new_node;
37      }
38    else
39      {
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

Stack | Heap

main

my_list	pointer to NODE NULL (0x0)

insert_node

head	pointer to NODE* ?
value	int ?
new_node	pointer to NODE ?

Note: ? refers to an uninitialized value

C/C++ details: none [default view]

5

그림으로 설명

C (C17 + GNU extensions)
known limitations

```
20        exit(EXIT_FAILURE);
21      }
22    new_node->data = value;
23    new_node->next = NULL;
24
25    return new_node;
26  }
27
28  // 연결 리스트에 노드를 추가하는 함수
29  void insert_node(NODE **head, int value)
30  {
31    NODE *new_node = create_node(value);
32
33    if(*head == NULL)
34      {
35        // 리스트가 비어있을 경우
36        *head = new_node;
37      }
38    else
39      {
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

Stack | Heap

main
- my_list: pointer to NODE, NULL (0x0)

insert_node
- head: pointer to NODE*
- value: int, 10
- new_node: pointer to NODE, ?

Note: ? refers to an uninitialized value

C/C++ details: none [default view]

6

그림으로 설명

C (C17 + GNU extensions)
known limitations

```
3  #include <stdlib.h>
4
5  // 연결 리스트의 노드를 나타내는 구조체
6  typedef struct node
7  {
8    int data;
9    struct node *next;
10 } NODE;
11
12 // 새로운 노드를 생성하는 함수
13 NODE *create_node(int value)
14 {
15   NODE *new_node = (NODE *)malloc(sizeof(NODE));
16
17   if(new_node == NULL)
18     {
19       printf("메모리 할당 오류₩n");
20       exit(EXIT_FAILURE);
21     }
22   new_node->data = value;
23   new_node->next = NULL;
24
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

Stack | Heap

main
- my_list: pointer to NODE, NULL (0x0)

insert_node
- head: pointer to NODE*
- value: int, 10
- new_node: pointer to NODE, ?

create_node
- value: int, ?
- new_node: pointer to NODE, ?

Note: ? refers to an uninitialized value

C/C++ details: none [default view]

7

그림으로 설명

C (C17 + GNU extensions)
known limitations

```
3  #include <stdlib.h>
4
5  // 연결 리스트의 노드를 나타내는 구조체
6  typedef struct node
7  {
8    int data;
9    struct node *next;
10 } NODE;
11
12 // 새로운 노드를 생성하는 함수
13 NODE *create_node(int value)
14 {
15   NODE *new_node = (NODE *)malloc(sizeof(NODE));
16
17   if(new_node == NULL)
18     {
19       printf("메모리 할당 오류₩n");
20       exit(EXIT_FAILURE);
21     }
22   new_node->data = value;
23   new_node->next = NULL;
24
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

Stack | Heap

main
- my_list: pointer to NODE, NULL (0x0)

insert_node
- head: pointer to NODE*
- value: int, 10
- new_node: pointer to NODE, ?

create_node
- value: int, 10
- new_node: pointer to NODE, ?

Note: ? refers to an uninitialized value

C/C++ details: none [default view]

8

그림으로 설명

C (C17 + GNU extensions)
known limitations

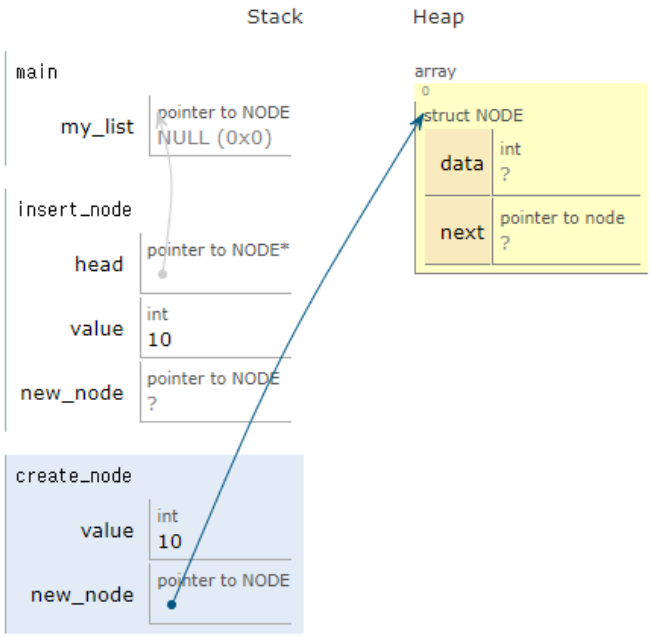
```c
#include <stdlib.h>

// 연결 리스트의 노드를 나타내는 구조체
typedef struct node
{
  int data;
  struct node *next;
} NODE;

// 새로운 노드를 생성하는 함수
NODE *create_node(int value)
{
  NODE *new_node = (NODE *)malloc(sizeof(NODE));

  if(new_node == NULL)
    {
      printf("메모리 할당 오류₩n");
      exit(EXIT_FAILURE);
    }
  new_node->data = value;
  new_node->next = NULL;
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: ? refers to an uninitialized value

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members.

9

그림으로 설명

C (C17 + GNU extensions)
known limitations

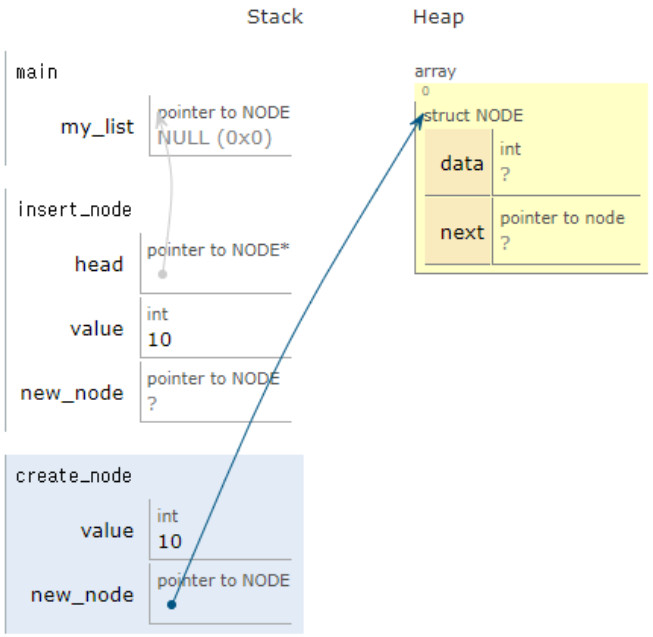
```
3  #include <stdlib.h>
4
5  // 연결 리스트의 노드를 나타내는 구조체
6  typedef struct node
7  {
8    int data;
9    struct node *next;
10 } NODE;
11
12 // 새로운 노드를 생성하는 함수
13 NODE *create_node(int value)
14 {
15   NODE *new_node = (NODE *)malloc(sizeof(NODE));
16
17   if(new_node == NULL)
18     {
19       printf("메모리 할당 오류₩n");
20       exit(EXIT_FAILURE);
21     }
22   new_node->data = value;
23   new_node->next = NULL;
24
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: ? refers to an uninitialized value

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members.

그림으로 설명

C (C17 + GNU extensions)
known limitations

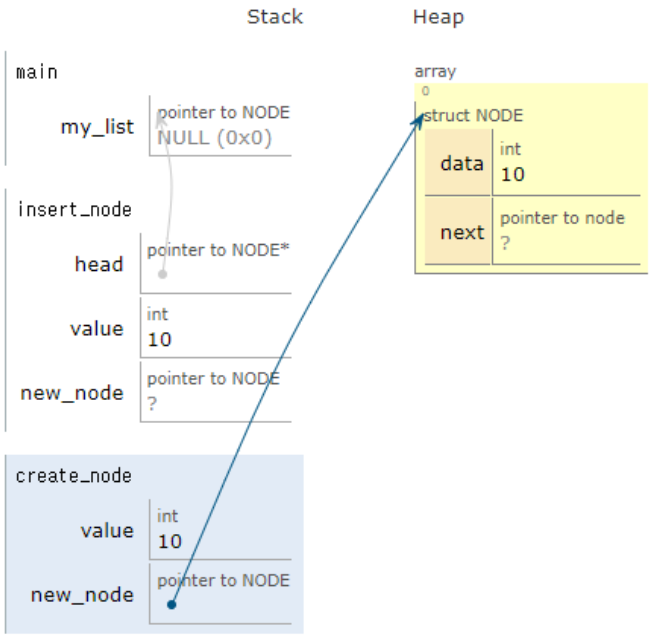
```
4
5  // 연결 리스트의 노드를 나타내는 구조체
6  typedef struct node
7  {
8    int data;
9    struct node *next;
10 } NODE;
11
12 // 새로운 노드를 생성하는 함수
13 NODE *create_node(int value)
14 {
15   NODE *new_node = (NODE *)malloc(sizeof(NODE));
16
17   if(new_node == NULL)
18     {
19       printf("메모리 할당 오류₩n");
20       exit(EXIT_FAILURE);
21     }
22   new_node->data = value;
23   new_node->next = NULL;
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: ? refers to an uninitialized value

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members.

그림으로 설명

C (C17 + GNU extensions)
known limitations

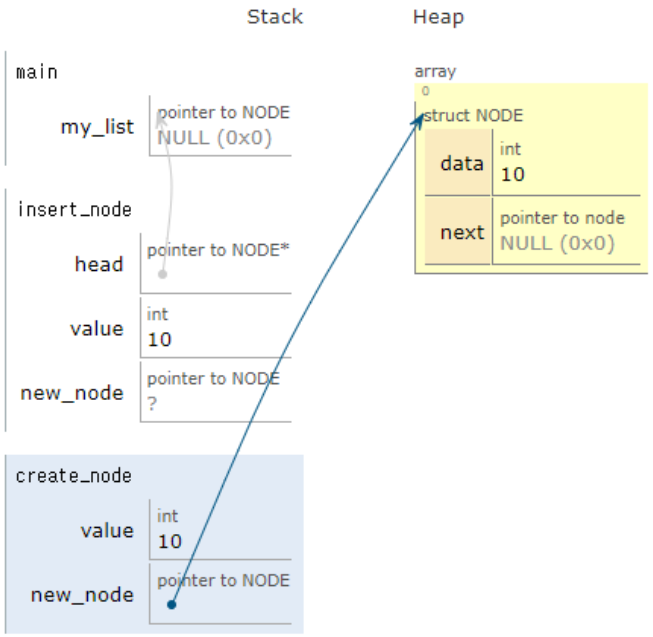
```
  NODE *new_node = (NODE *)malloc(sizeof(NODE));

  if(new_node == NULL)
    {
      printf("메모리 할당 오류₩n");
      exit(EXIT_FAILURE);
    }
  new_node->data = value;
  new_node->next = NULL;

  return new_node;
}

// 연결 리스트에 노드를 추가하는 함수
void insert_node(NODE **head, int value)
{
  NODE *new_node = create_node(value);

  if(*head == NULL)
    {
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: ? refers to an uninitialized value

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members.

그림으로 설명

C (C17 + GNU extensions)
known limitations

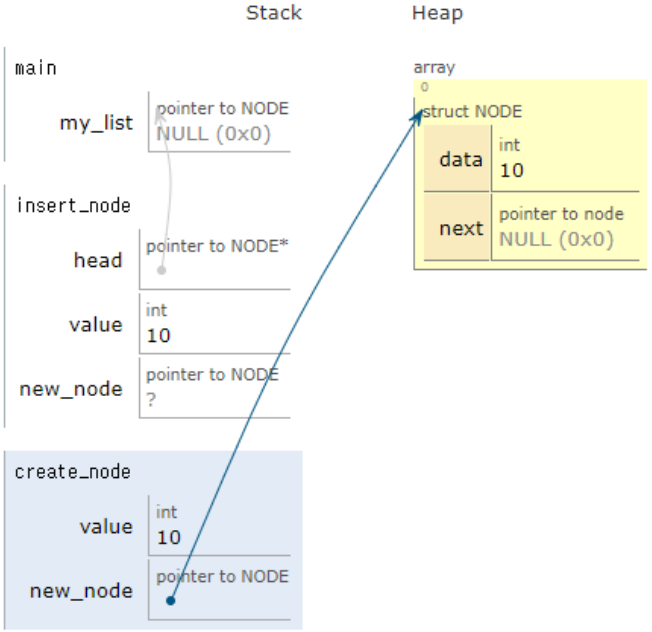
```
  NODE *new_node = (NODE *)malloc(sizeof(NODE));

  if(new_node == NULL)
    {
      printf("메모리 할당 오류₩n");
      exit(EXIT_FAILURE);
    }
  new_node->data = value;
  new_node->next = NULL;

  return new_node;
}

// 연결 리스트에 노드를 추가하는 함수
void insert_node(NODE **head, int value)
{
  NODE *new_node = create_node(value);

  if(*head == NULL)
    {
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: ? refers to an uninitialized value

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members.

그림으로 설명

C (C17 + GNU extensions)
known limitations

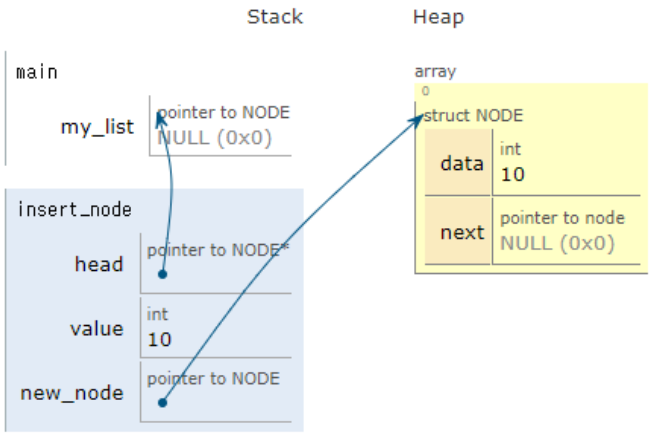
```
15    NODE *new_node = (NODE *)malloc(sizeof(NODE));
16
17    if(new_node == NULL)
18      {
19        printf("메모리 할당 오류₩n");
20        exit(EXIT_FAILURE);
21      }
22    new_node->data = value;
23    new_node->next = NULL;
24
25    return new_node;
26  }
27
28  // 연결 리스트에 노드를 추가하는 함수
29  void insert_node(NODE **head, int value)
30  {
31    NODE *new_node = create_node(value);
32
33    if(*head == NULL)
34      {
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

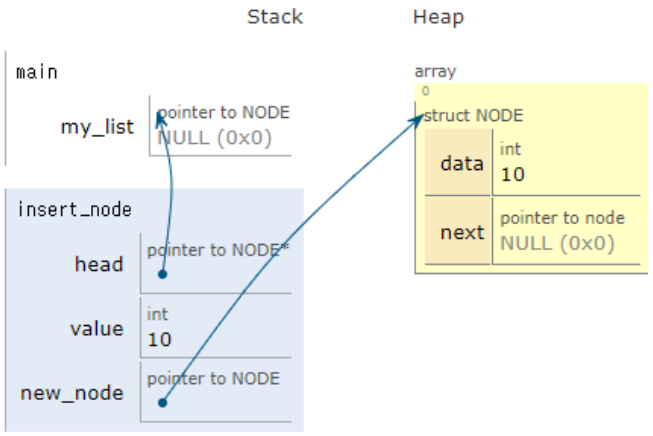
```
25    return new_node;
26  }
27
28  // 연결 리스트에 노드를 추가하는 함수
29  void insert_node(NODE **head, int value)
30  {
31    NODE *new_node = create_node(value);
32
33    if(*head == NULL)
34      {
35        // 리스트가 비어있을 경우
36        *head = new_node;
37      }
38    else
39      {
40        // 리스트가 비어있지 않을 경우
41        NODE *temp = *head;
42        while(temp->next != NULL)
43          {
44            temp = temp->next;
45          }
46        temp->next = new_node;
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

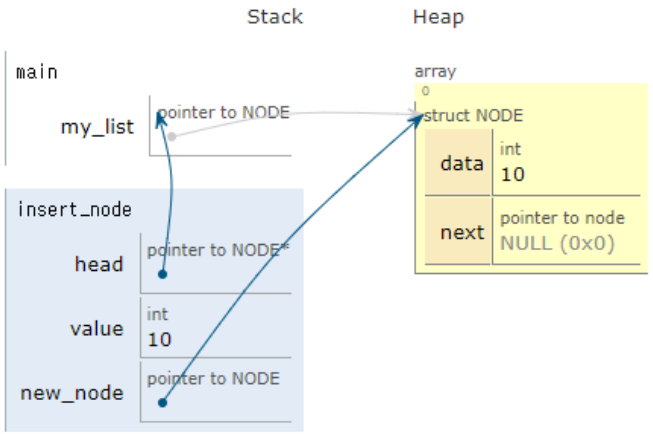
```
  else
    {
      // 리스트가 비어있지 않을 경우
      NODE *temp = *head;
      while(temp->next != NULL)
        {
          temp = temp->next;
        }
      temp->next = new_node;
    }
}

// 연결 리스트의 노드를 출력하는 함수
void print_list(NODE *head)
{
  NODE *temp = head;
  while(temp != NULL)
    {
      printf("%d -> ", temp->data);
      temp = temp->next;
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

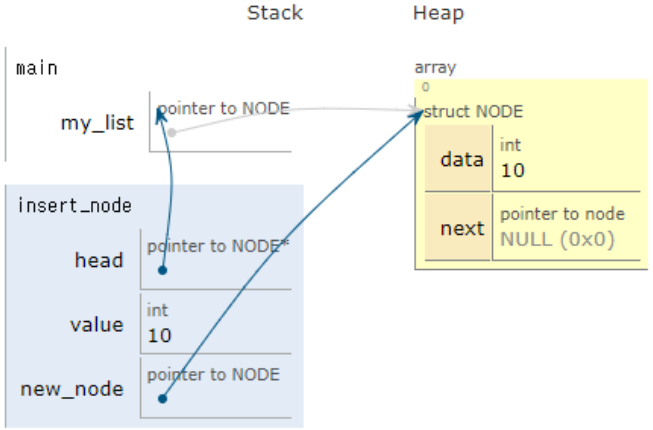
```
  else
    {
      // 리스트가 비어있지 않을 경우
      NODE *temp = *head;
      while(temp->next != NULL)
        {
          temp = temp->next;
        }
      temp->next = new_node;
    }
}

// 연결 리스트의 노드를 출력하는 함수
void print_list(NODE *head)
{
  NODE *temp = head;
  while(temp != NULL)
    {
      printf("%d -> ", temp->data);
      temp = temp->next;
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

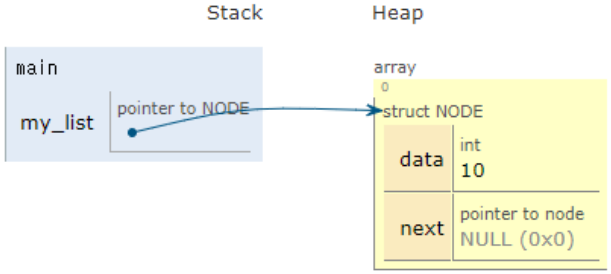
```
    }
  printf("메모리 해제 완료₩n");
}

int main()
{
  NODE *my_list = NULL;

  // 연결 리스트에 노드 추가
  insert_node(&my_list, 10);
  insert_node(&my_list, 20);
  insert_node(&my_list, 30);

  // 연결 리스트 출력
  printf("연결 리스트: ");
  print_list(my_list);

  // 메모리 해제
  free_list(my_list);
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

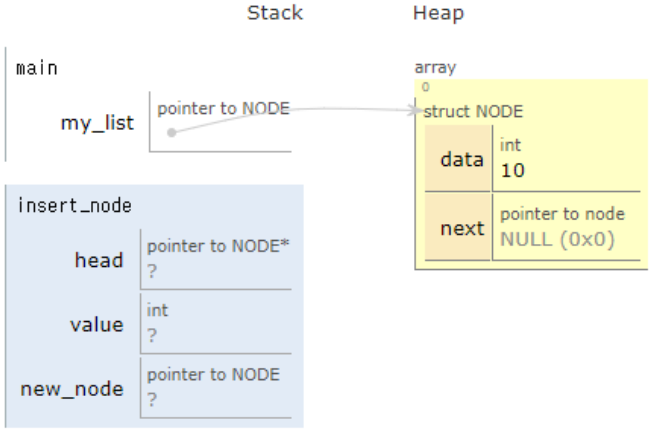
```
20        exit(EXIT_FAILURE);
21      }
22    new_node->data = value;
23    new_node->next = NULL;
24
25    return new_node;
26  }
27
28  // 연결 리스트에 노드를 추가하는 함수
29  void insert_node(NODE **head, int value)
30  {
31    NODE *new_node = create_node(value);
32
33    if(*head == NULL)
34      {
35        // 리스트가 비어있을 경우
36        *head = new_node;
37      }
38    else
39      {
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: ? refers to an uninitialized value

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

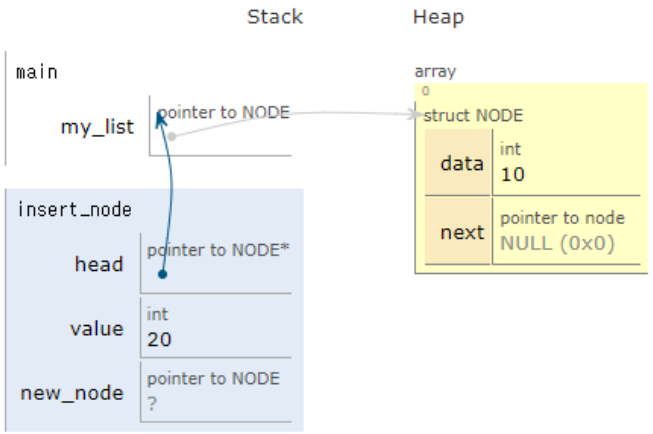
```
20        exit(EXIT_FAILURE);
21      }
22    new_node->data = value;
23    new_node->next = NULL;
24
25    return new_node;
26  }
27
28  // 연결 리스트에 노드를 추가하는 함수
29  void insert_node(NODE **head, int value)
30  {
31    NODE *new_node = create_node(value);
32
33    if(*head == NULL)
34      {
35        // 리스트가 비어있을 경우
36        *head = new_node;
37      }
38    else
39      {
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: ? refers to an uninitialized value

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

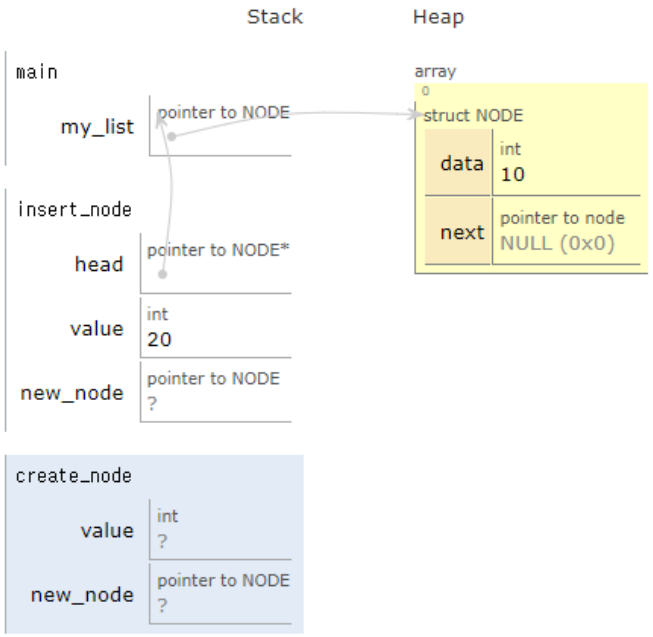
```
3  #include <stdlib.h>
4
5  // 연결 리스트의 노드를 나타내는 구조체
6  typedef struct node
7  {
8    int data;
9    struct node *next;
10 } NODE;
11
12 // 새로운 노드를 생성하는 함수
13 NODE *create_node(int value)
14 {
15   NODE *new_node = (NODE *)malloc(sizeof(NODE));
16
17   if(new_node == NULL)
18     {
19       printf("메모리 할당 오류₩n");
20       exit(EXIT_FAILURE);
21     }
22   new_node->data = value;
23   new_node->next = NULL;
24
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: ? refers to an uninitialized value

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members.

그림으로 설명

C (C17 + GNU extensions)
known limitations

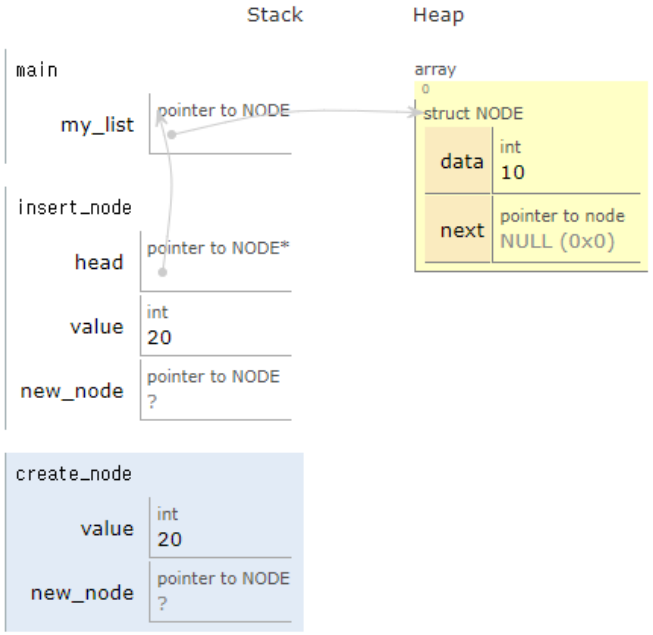
```
#include <stdlib.h>

// 연결 리스트의 노드를 나타내는 구조체
typedef struct node
{
  int data;
  struct node *next;
} NODE;

// 새로운 노드를 생성하는 함수
NODE *create_node(int value)
{
  NODE *new_node = (NODE *)malloc(sizeof(NODE));

  if(new_node == NULL)
    {
      printf("메모리 할당 오류₩n");
      exit(EXIT_FAILURE);
    }
  new_node->data = value;
  new_node->next = NULL;
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: ? refers to an uninitialized value

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members.

그림으로 설명

C (C17 + GNU extensions)
known limitations

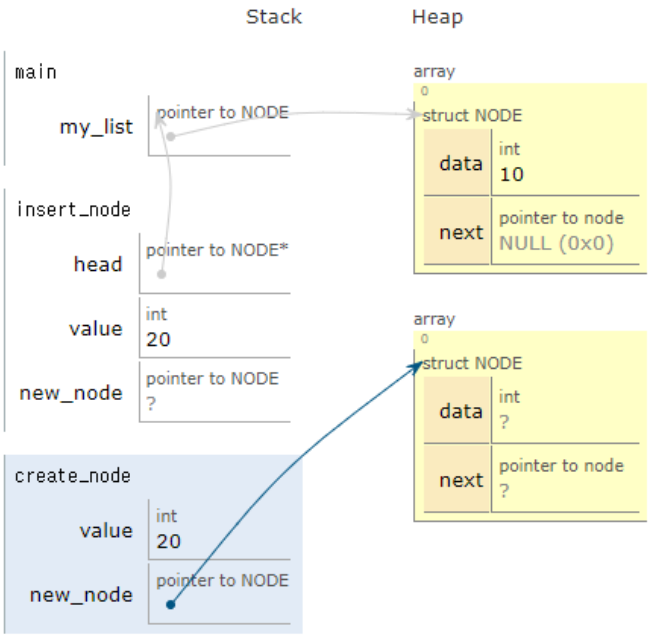
```
3  #include <stdlib.h>
4
5  // 연결 리스트의 노드를 나타내는 구조체
6  typedef struct node
7  {
8    int data;
9    struct node *next;
10 } NODE;
11
12 // 새로운 노드를 생성하는 함수
13 NODE *create_node(int value)
14 {
15   NODE *new_node = (NODE *)malloc(sizeof(NODE));
16
17   if(new_node == NULL)
18     {
19       printf("메모리 할당 오류₩n");
20       exit(EXIT_FAILURE);
21     }
22   new_node->data = value;
23   new_node->next = NULL;
24
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: ? refers to an uninitialized value

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members.

그림으로 설명

C (C17 + GNU extensions)
known limitations

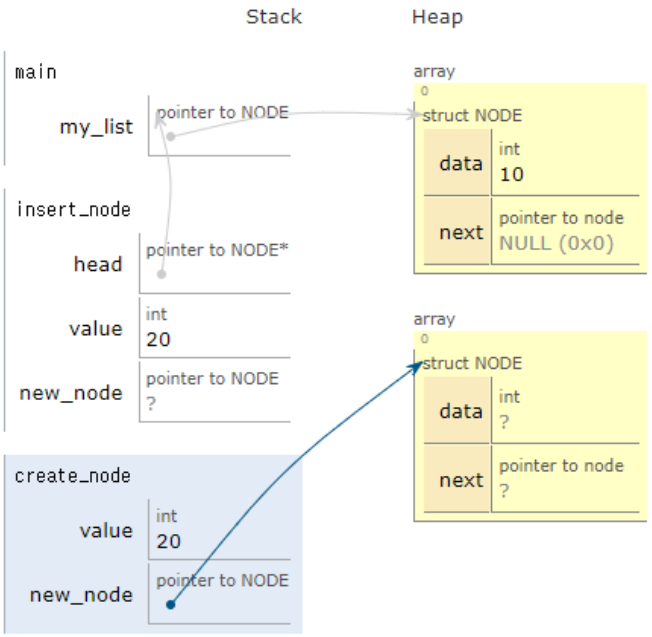
```
3  #include <stdlib.h>
4
5  // 연결 리스트의 노드를 나타내는 구조체
6  typedef struct node
7  {
8    int data;
9    struct node *next;
10 } NODE;
11
12 // 새로운 노드를 생성하는 함수
13 NODE *create_node(int value)
14 {
15   NODE *new_node = (NODE *)malloc(sizeof(NODE));
16
17   if(new_node == NULL)
18     {
19       printf("메모리 할당 오류₩n");
20       exit(EXIT_FAILURE);
21     }
22   new_node->data = value;
23   new_node->next = NULL;
24
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: ? refers to an uninitialized value

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members.

그림으로 설명

C (C17 + GNU extensions)
known limitations

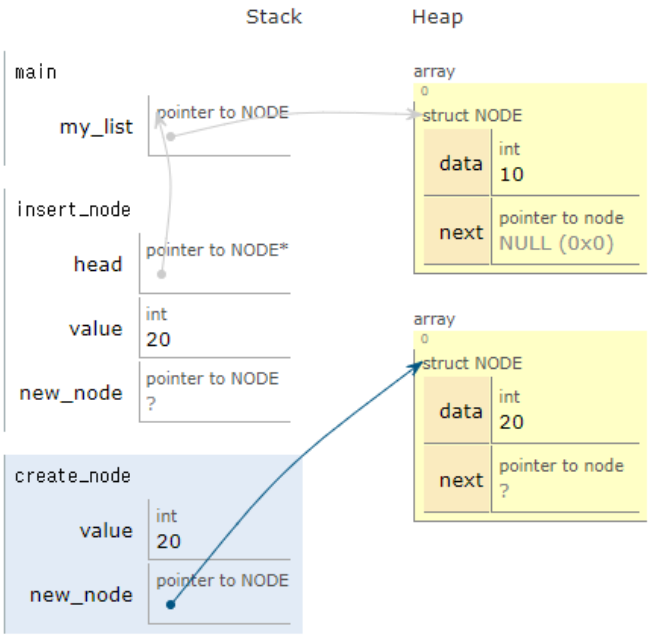
```
3  #include <stdlib.h>
4
5  // 연결 리스트의 노드를 나타내는 구조체
6  typedef struct node
7  {
8    int data;
9    struct node *next;
10 } NODE;
11
12 // 새로운 노드를 생성하는 함수
13 NODE *create_node(int value)
14 {
15   NODE *new_node = (NODE *)malloc(sizeof(NODE));
16
17   if(new_node == NULL)
18     {
19       printf("메모리 할당 오류₩n");
20       exit(EXIT_FAILURE);
21     }
22   new_node->data = value;
23   new_node->next = NULL;
24
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: ? refers to an uninitialized value

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members.

그림으로 설명

C (C17 + GNU extensions)
known limitations

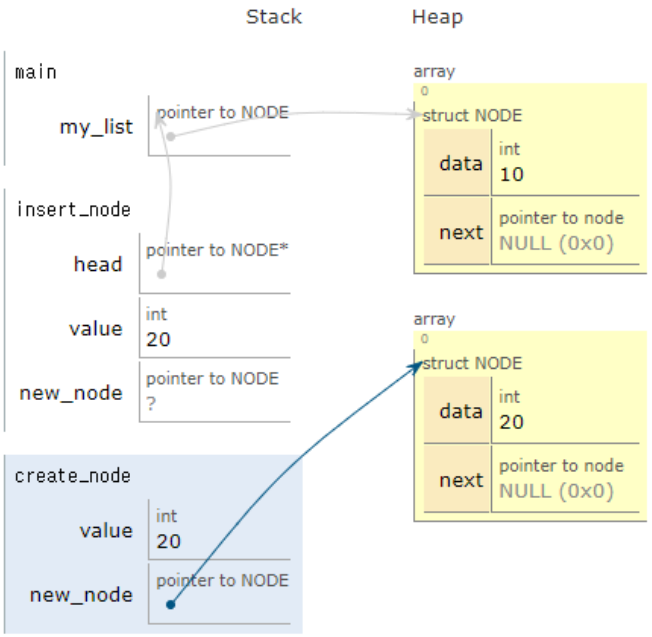
```
  NODE *new_node = (NODE *)malloc(sizeof(NODE));

  if(new_node == NULL)
    {
      printf("메모리 할당 오류₩n");
      exit(EXIT_FAILURE);
    }
  new_node->data = value;
  new_node->next = NULL;

  return new_node;
}

// 연결 리스트에 노드를 추가하는 함수
void insert_node(NODE **head, int value)
{
  NODE *new_node = create_node(value);

  if(*head == NULL)
    {
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: ? refers to an uninitialized value

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members.

그림으로 설명

C (C17 + GNU extensions)
known limitations

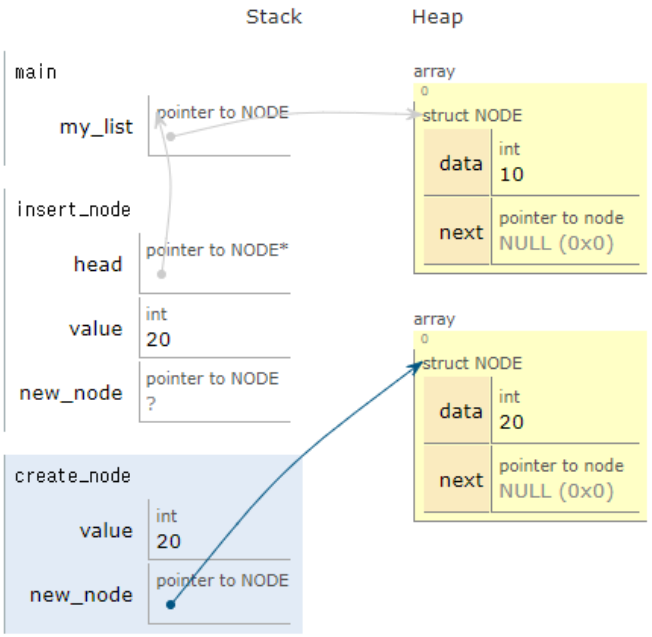
```
  NODE *new_node = (NODE *)malloc(sizeof(NODE));

  if(new_node == NULL)
    {
      printf("메모리 할당 오류₩n");
      exit(EXIT_FAILURE);
    }
  new_node->data = value;
  new_node->next = NULL;

  return new_node;
}

// 연결 리스트에 노드를 추가하는 함수
void insert_node(NODE **head, int value)
{
  NODE *new_node = create_node(value);

  if(*head == NULL)
    {
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: ? refers to an uninitialized value

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members.

그림으로 설명

C (C17 + GNU extensions)
known limitations

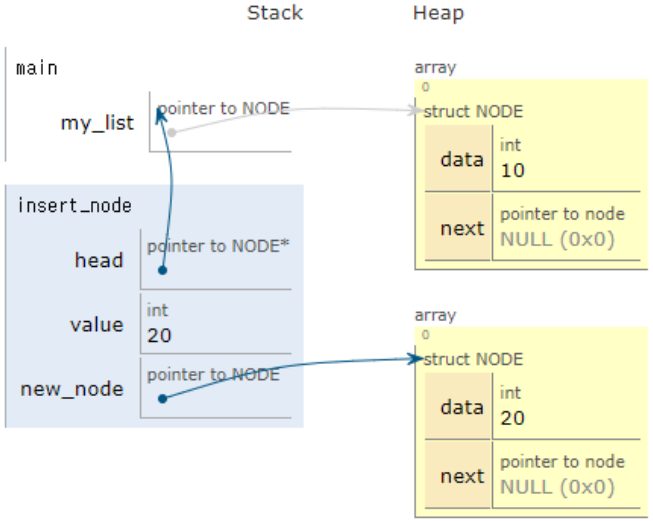
```
15    NODE *new_node = (NODE *)malloc(sizeof(NODE));
16
17    if(new_node == NULL)
18      {
19        printf("메모리 할당 오류₩n");
20        exit(EXIT_FAILURE);
21      }
22    new_node->data = value;
23    new_node->next = NULL;
24
25    return new_node;
26  }
27
28  // 연결 리스트에 노드를 추가하는 함수
29  void insert_node(NODE **head, int value)
30  {
31    NODE *new_node = create_node(value);
32
33    if(*head == NULL)
34      {
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

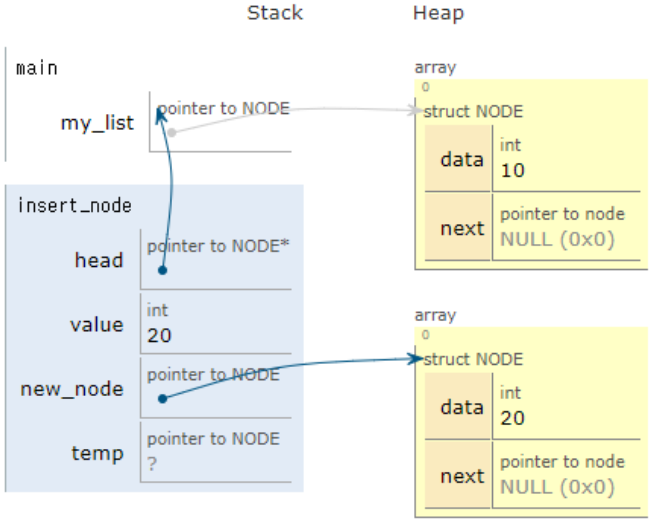
```
{
  NODE *new_node = create_node(value);

  if(*head == NULL)
    {
      // 리스트가 비어있을 경우
      *head = new_node;
    }
  else
    {
      // 리스트가 비어있지 않을 경우
      NODE *temp = *head;
      while(temp->next != NULL)
        {
          temp = temp->next;
        }
      temp->next = new_node;
    }
}

// 연결 리스트의 노드를 출력하는 함수
void print_list(NODE *head)
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: ? refers to an uninitialized value

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

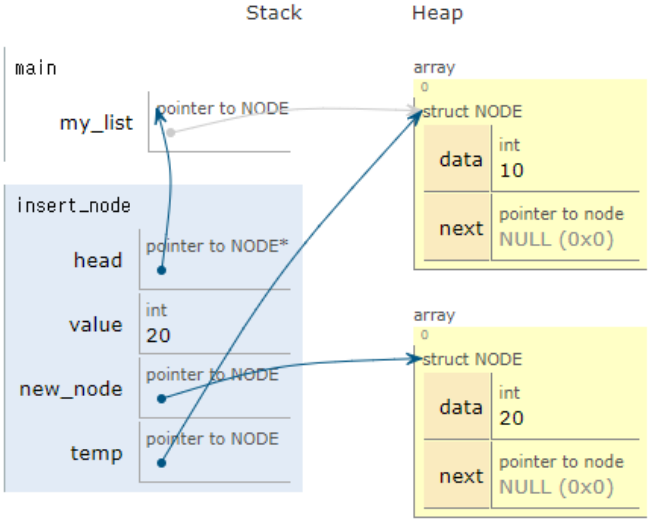
```
31    NODE *new_node = create_node(value);
32
33    if(*head == NULL)
34      {
35        // 리스트가 비어있을 경우
36        *head = new_node;
37      }
38    else
39      {
40        // 리스트가 비어있지 않을 경우
41        NODE *temp = *head;
42        while(temp->next != NULL)
43          {
44            temp = temp->next;
45          }
46        temp->next = new_node;
47      }
48  }
49
50  // 연결 리스트의 노드를 출력하는 함수
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

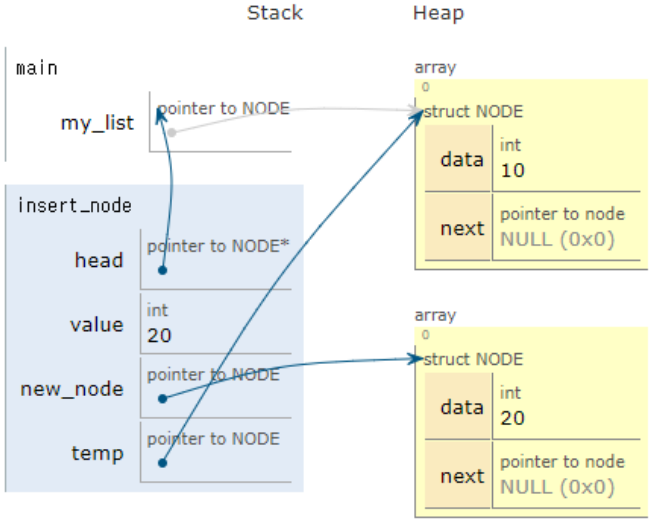
```
{
  NODE *new_node = create_node(value);

  if(*head == NULL)
    {
      // 리스트가 비어있을 경우
      *head = new_node;
    }
  else
    {
      // 리스트가 비어있지 않을 경우
      NODE *temp = *head;
      while(temp->next != NULL)
        {
          temp = temp->next;
        }
      temp->next = new_node;
    }
}

// 연결 리스트의 노드를 출력하는 함수
void print_list(NODE *head)
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

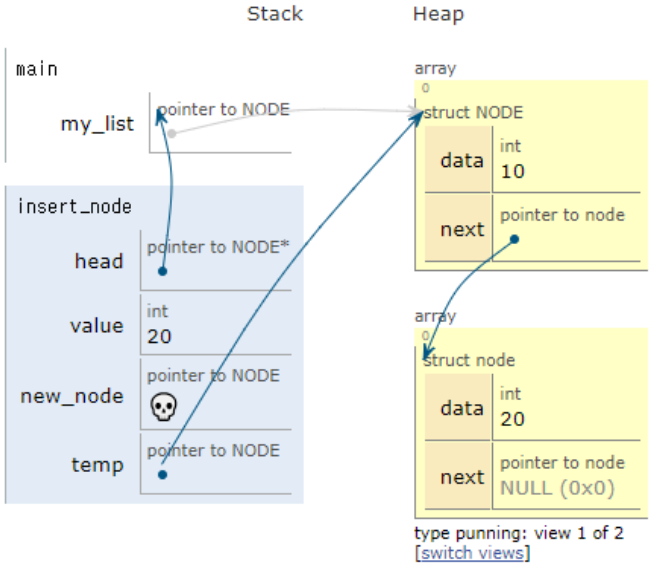
```
{
  NODE *new_node = create_node(value);

  if(*head == NULL)
    {
      // 리스트가 비어있을 경우
      *head = new_node;
    }
  else
    {
      // 리스트가 비어있지 않을 경우
      NODE *temp = *head;
      while(temp->next != NULL)
        {
          temp = temp->next;
        }
      temp->next = new_node;
    }
}

// 연결 리스트의 노드를 출력하는 함수
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



type punning: view 1 of 2
[switch views]

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

Note: 💀 means a pointer points to memory that is either unallocated or misaligned with data boundaries. 💀 locations are approximate and may not match the pointer's real address

그림으로 설명

C (C17 + GNU extensions)
known limitations

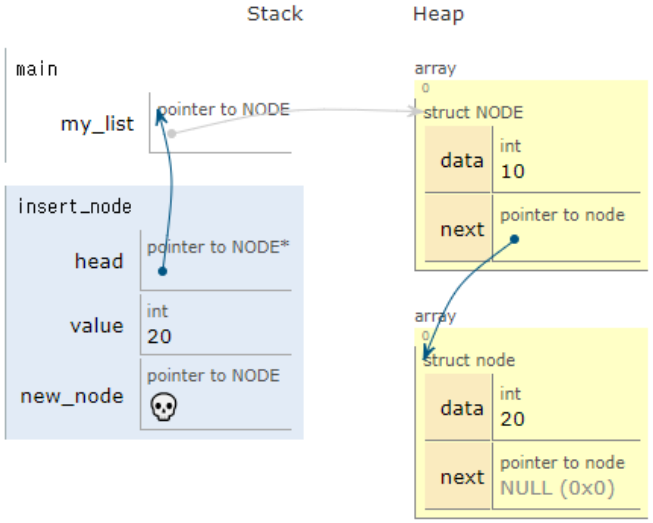
```
{
  NODE *new_node = create_node(value);

  if(*head == NULL)
    {
      // 리스트가 비어있을 경우
      *head = new_node;
    }
  else
    {
      // 리스트가 비어있지 않을 경우
      NODE *temp = *head;
      while(temp->next != NULL)
        {
          temp = temp->next;
        }
      temp->next = new_node;
    }
}

// 연결 리스트의 노드를 출력하는 함수
void print_list(NODE *head)
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



type punning: view 1 of 2
[switch views]

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

Note: 💀 means a pointer points to memory that is either unallocated or misaligned with data boundaries. 💀 locations are approximate and may not match the pointer's real address

그림으로 설명

C (C17 + GNU extensions)
known limitations

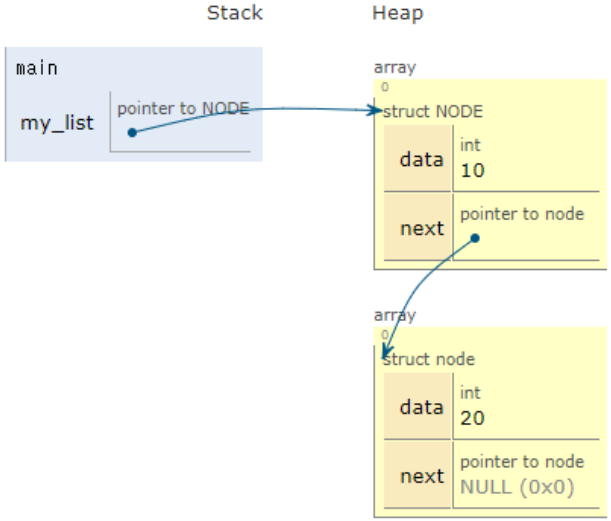
```
  printf("메모리 해제 완료₩n");
}

int main()
{
  NODE *my_list = NULL;

  // 연결 리스트에 노드 추가
  insert_node(&my_list, 10);
  insert_node(&my_list, 20);
  insert_node(&my_list, 30);

  // 연결 리스트 출력
  printf("연결 리스트: ");
  print_list(my_list);

  // 메모리 해제
  free_list(my_list);

  return 0;
}
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

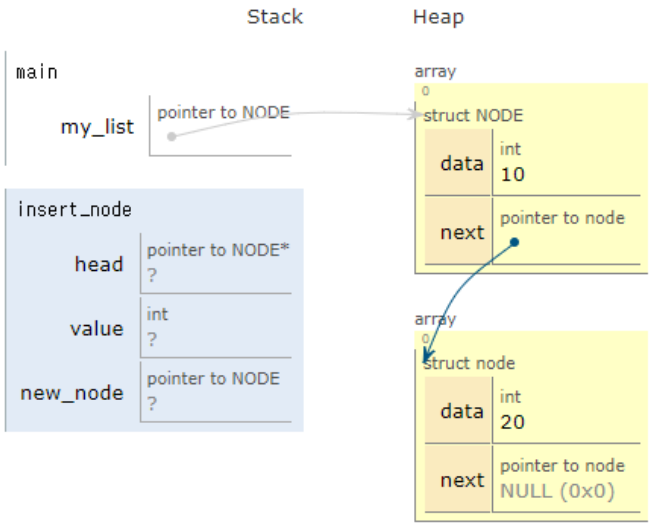
```
20        exit(EXIT_FAILURE);
21      }
22    new_node->data = value;
23    new_node->next = NULL;
24
25    return new_node;
26  }
27
28  // 연결 리스트에 노드를 추가하는 함수
29  void insert_node(NODE **head, int value)
30  {
31    NODE *new_node = create_node(value);
32
33    if(*head == NULL)
34      {
35        // 리스트가 비어있을 경우
36        *head = new_node;
37      }
38    else
39      {
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: ? refers to an uninitialized value

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

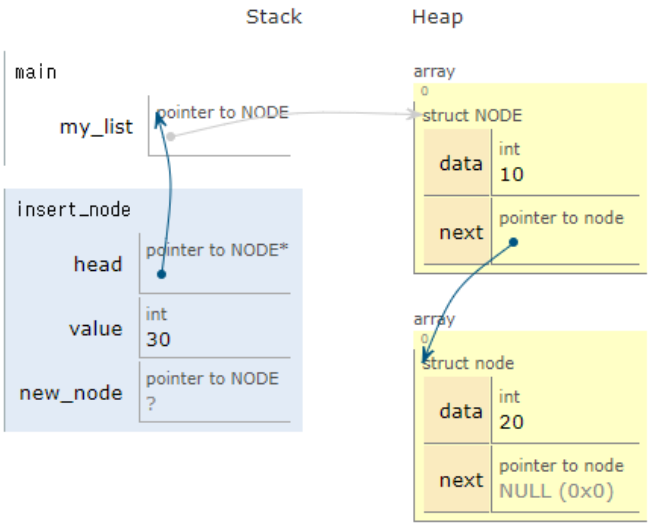
```
20        exit(EXIT_FAILURE);
21      }
22    new_node->data = value;
23    new_node->next = NULL;
24
25    return new_node;
26  }
27
28  // 연결 리스트에 노드를 추가하는 함수
29  void insert_node(NODE **head, int value)
30  {
31    NODE *new_node = create_node(value);
32
33    if(*head == NULL)
34      {
35        // 리스트가 비어있을 경우
36        *head = new_node;
37      }
38    else
39      {
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: ? refers to an uninitialized value

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

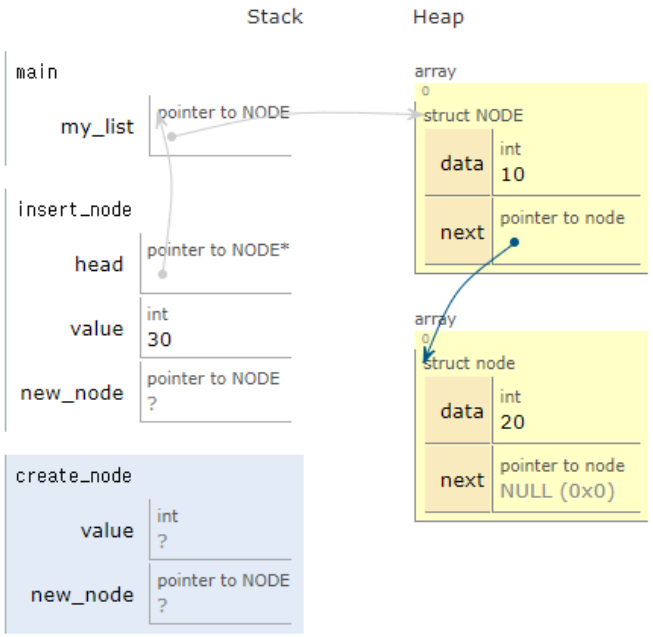
```
#include <stdlib.h>

// 연결 리스트의 노드를 나타내는 구조체
typedef struct node
{
  int data;
  struct node *next;
} NODE;

// 새로운 노드를 생성하는 함수
NODE *create_node(int value)
{
  NODE *new_node = (NODE *)malloc(sizeof(NODE));

  if(new_node == NULL)
    {
      printf("메모리 할당 오류₩n");
      exit(EXIT_FAILURE);
    }
  new_node->data = value;
  new_node->next = NULL;
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: ? refers to an uninitialized value

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members.

그림으로 설명

C (C17 + GNU extensions)
known limitations

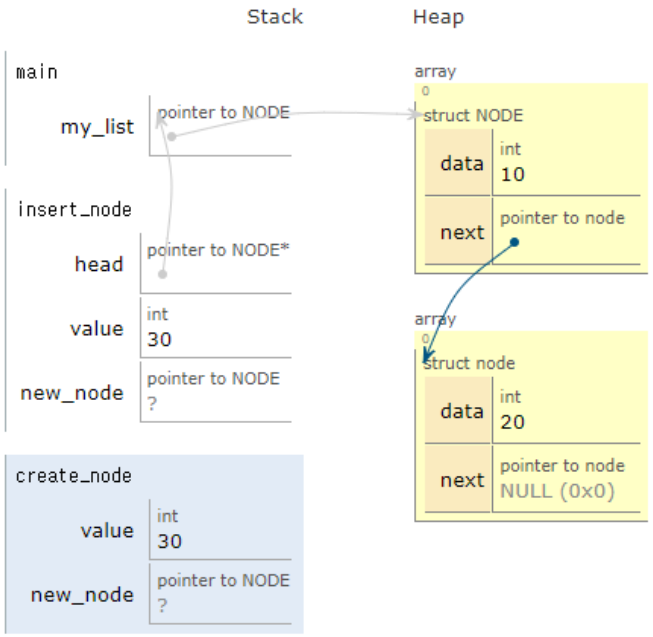
```
3  #include <stdlib.h>
4
5  // 연결 리스트의 노드를 나타내는 구조체
6  typedef struct node
7  {
8    int data;
9    struct node *next;
10 } NODE;
11
12 // 새로운 노드를 생성하는 함수
13 NODE *create_node(int value)
14 {
15   NODE *new_node = (NODE *)malloc(sizeof(NODE));
16
17   if(new_node == NULL)
18     {
19       printf("메모리 할당 오류₩n");
20       exit(EXIT_FAILURE);
21     }
22   new_node->data = value;
23   new_node->next = NULL;
24
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



Note: ? refers to an uninitialized value

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members.

그림으로 설명

C (C17 + GNU extensions)
known limitations

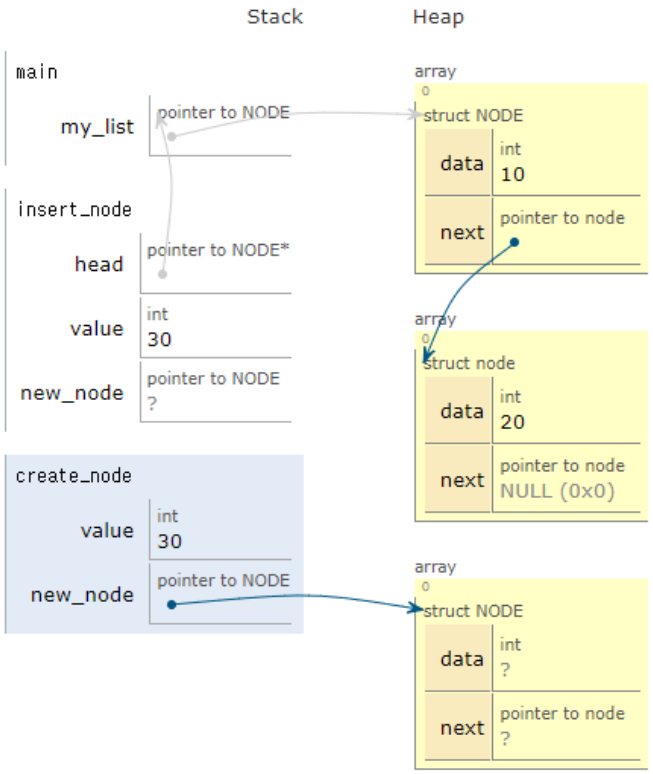
```
#include <stdlib.h>

// 연결 리스트의 노드를 나타내는 구조체
typedef struct node
{
  int data;
  struct node *next;
} NODE;

// 새로운 노드를 생성하는 함수
NODE *create_node(int value)
{
  NODE *new_node = (NODE *)malloc(sizeof(NODE));

  if(new_node == NULL)
    {
      printf("메모리 할당 오류₩n");
      exit(EXIT_FAILURE);
    }
  new_node->data = value;
  new_node->next = NULL;
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



그림으로 설명

C (C17 + GNU extensions)
known limitations

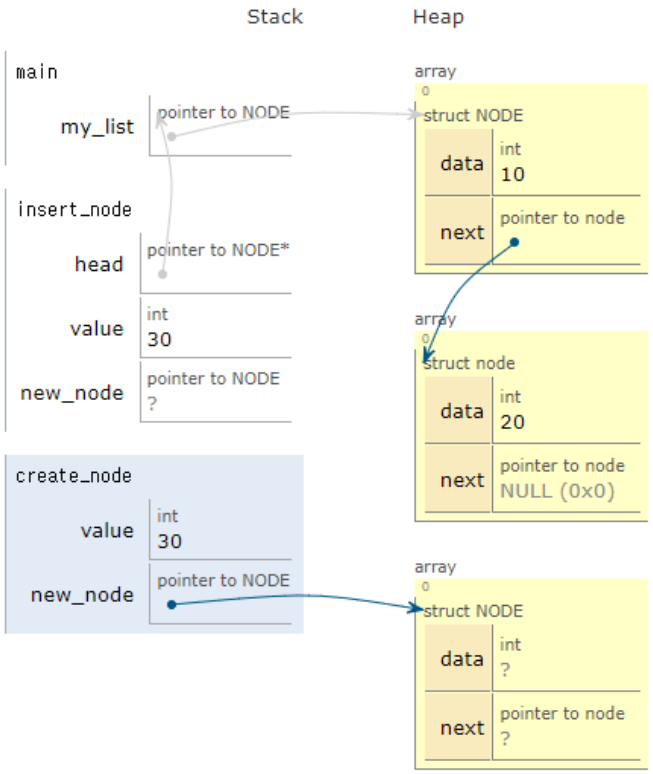
```
3  #include <stdlib.h>
4
5  // 연결 리스트의 노드를 나타내는 구조체
6  typedef struct node
7  {
8    int data;
9    struct node *next;
10 } NODE;
11
12 // 새로운 노드를 생성하는 함수
13 NODE *create_node(int value)
14 {
15   NODE *new_node = (NODE *)malloc(sizeof(NODE));
16
17   if(new_node == NULL)
18     {
19       printf("메모리 할당 오류₩n");
20       exit(EXIT_FAILURE);
21     }
22   new_node->data = value;
23   new_node->next = NULL;
24
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



그림으로 설명

C (C17 + GNU extensions)
known limitations

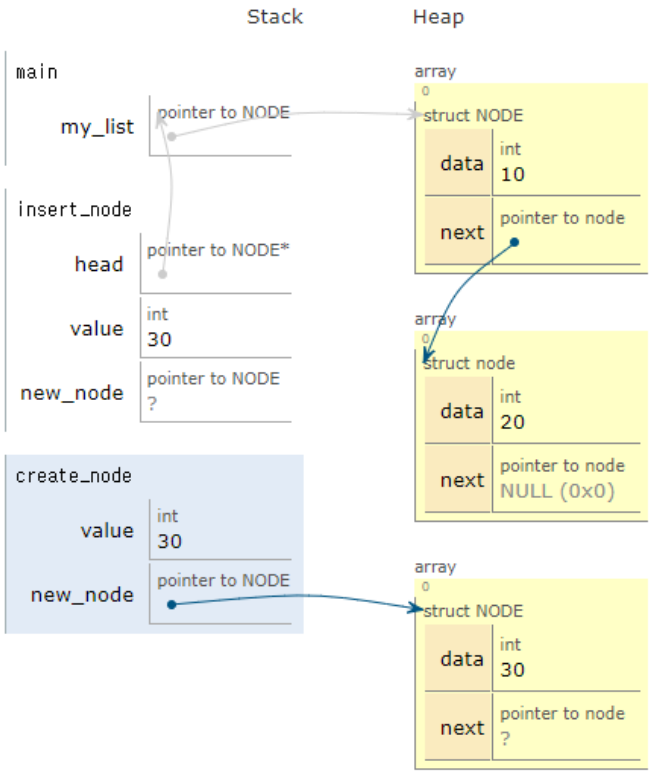
```
#include <stdlib.h>

// 연결 리스트의 노드를 나타내는 구조체
typedef struct node
{
  int data;
  struct node *next;
} NODE;

// 새로운 노드를 생성하는 함수
NODE *create_node(int value)
{
  NODE *new_node = (NODE *)malloc(sizeof(NODE));

  if(new_node == NULL)
    {
      printf("메모리 할당 오류₩n");
      exit(EXIT_FAILURE);
    }
  new_node->data = value;
  new_node->next = NULL;
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



그림으로 설명

C (C17 + GNU extensions)
known limitations

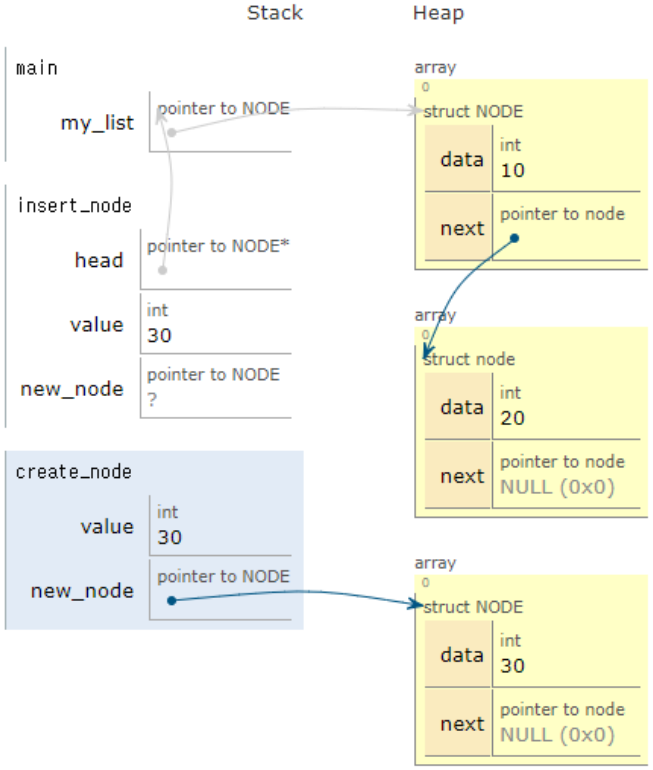
```
15    NODE *new_node = (NODE *)malloc(sizeof(NODE));
16
17    if(new_node == NULL)
18      {
19        printf("메모리 할당 오류₩n");
20        exit(EXIT_FAILURE);
21      }
22    new_node->data = value;
23    new_node->next = NULL;
24
25    return new_node;
26  }
27
28  // 연결 리스트에 노드를 추가하는 함수
29  void insert_node(NODE **head, int value)
30  {
31    NODE *new_node = create_node(value);
32
33    if(*head == NULL)
34      {
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



그림으로 설명

C (C17 + GNU extensions)
known limitations

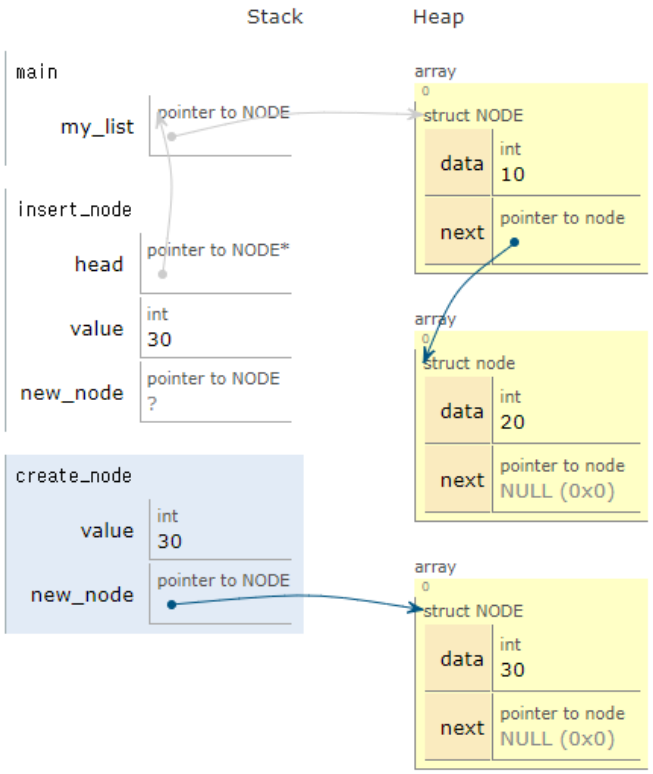
```
15    NODE *new_node = (NODE *)malloc(sizeof(NODE));
16
17    if(new_node == NULL)
18      {
19        printf("메모리 할당 오류₩n");
20        exit(EXIT_FAILURE);
21      }
22    new_node->data = value;
23    new_node->next = NULL;
24
25    return new_node;
26  }
27
28  // 연결 리스트에 노드를 추가하는 함수
29  void insert_node(NODE **head, int value)
30  {
31    NODE *new_node = create_node(value);
32
33    if(*head == NULL)
34      {
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



그림으로 설명

C (C17 + GNU extensions)
known limitations

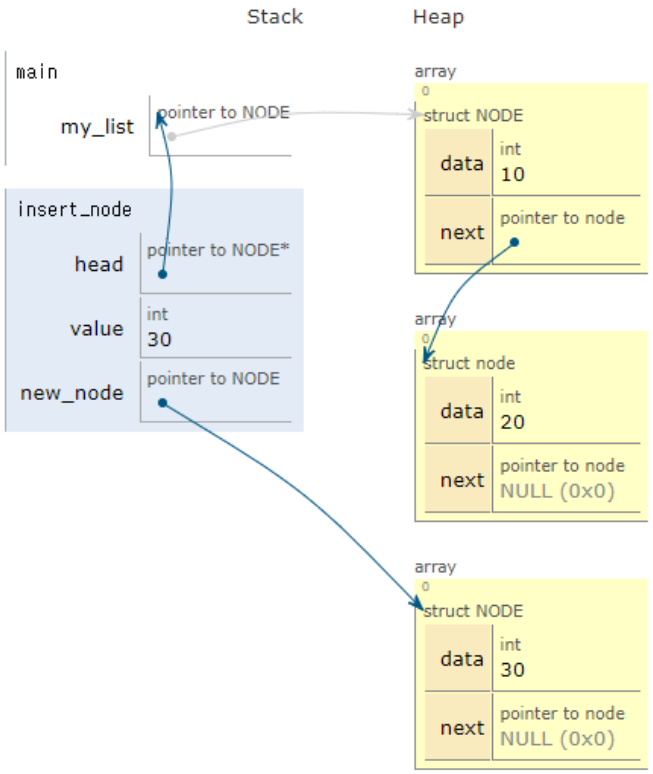
```
15   NODE *new_node = (NODE *)malloc(sizeof(NODE));
16
17   if(new_node == NULL)
18     {
19       printf("메모리 할당 오류₩n");
20       exit(EXIT_FAILURE);
21     }
22   new_node->data = value;
23   new_node->next = NULL;
24
25   return new_node;
26 }
27
28 // 연결 리스트에 노드를 추가하는 함수
29 void insert_node(NODE **head, int value)
30 {
31   NODE *new_node = create_node(value);
32
33   if(*head == NULL)
34     {
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



그림으로 설명

C (C17 + GNU extensions)
known limitations

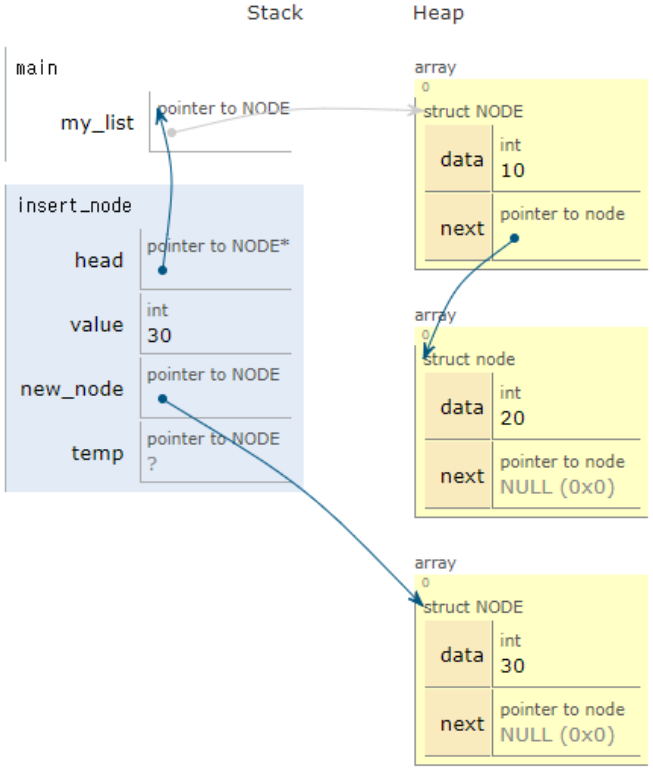
```
31    NODE *new_node = create_node(value);
32
33    if(*head == NULL)
34      {
35        // 리스트가 비어있을 경우
36        *head = new_node;
37      }
38    else
39      {
40        // 리스트가 비어있지 않을 경우
41        NODE *temp = *head;
42        while(temp->next != NULL)
43          {
44            temp = temp->next;
45          }
46        temp->next = new_node;
47      }
48  }
49
50  // 연결 리스트의 노드를 출력하는 함수
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



그림으로 설명

C (C17 + GNU extensions)
known limitations

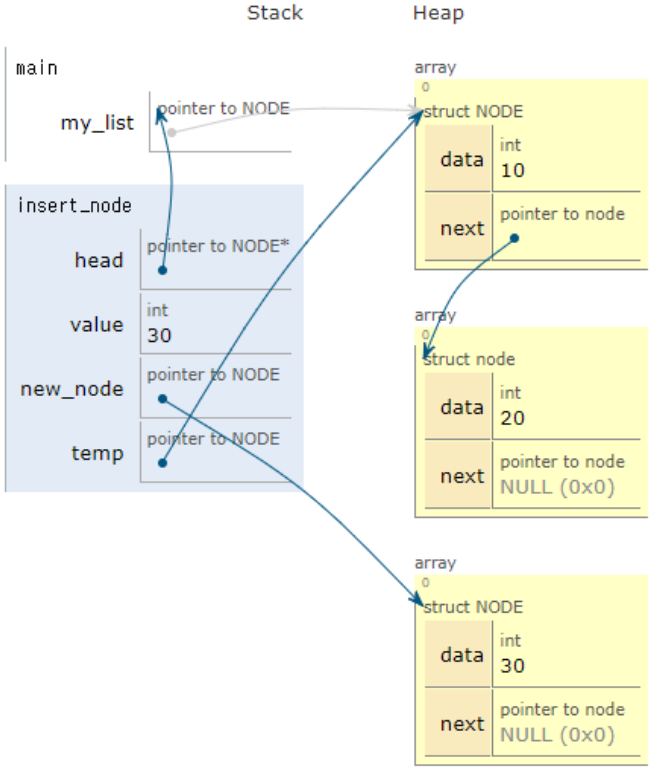
```
{
  NODE *new_node = create_node(value);

  if(*head == NULL)
    {
      // 리스트가 비어있을 경우
      *head = new_node;
    }
  else
    {
      // 리스트가 비어있지 않을 경우
      NODE *temp = *head;
      while(temp->next != NULL)
        {
          temp = temp->next;
        }
      temp->next = new_node;
    }
}

// 연결 리스트의 노드를 출력하는 함수
void print_list(NODE *head)
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



그림으로 설명

C (C17 + GNU extensions)
known limitations

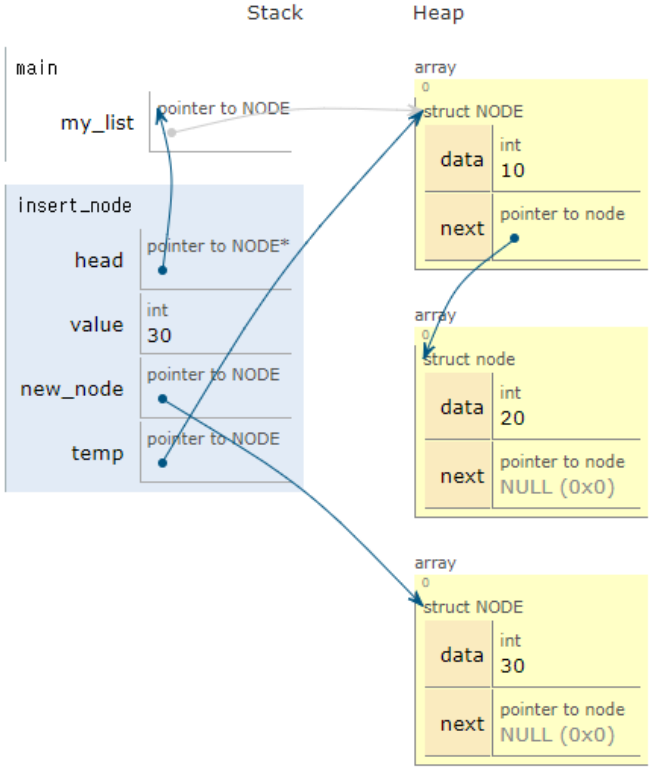
```
{
  NODE *new_node = create_node(value);

  if(*head == NULL)
    {
      // 리스트가 비어있을 경우
      *head = new_node;
    }
  else
    {
      // 리스트가 비어있지 않을 경우
      NODE *temp = *head;
      while(temp->next != NULL)
        {
          temp = temp->next;
        }
      temp->next = new_node;
    }
}

// 연결 리스트의 노드를 출력하는 함수
void print_list(NODE *head)
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



그림으로 설명

C (C17 + GNU extensions)
known limitations

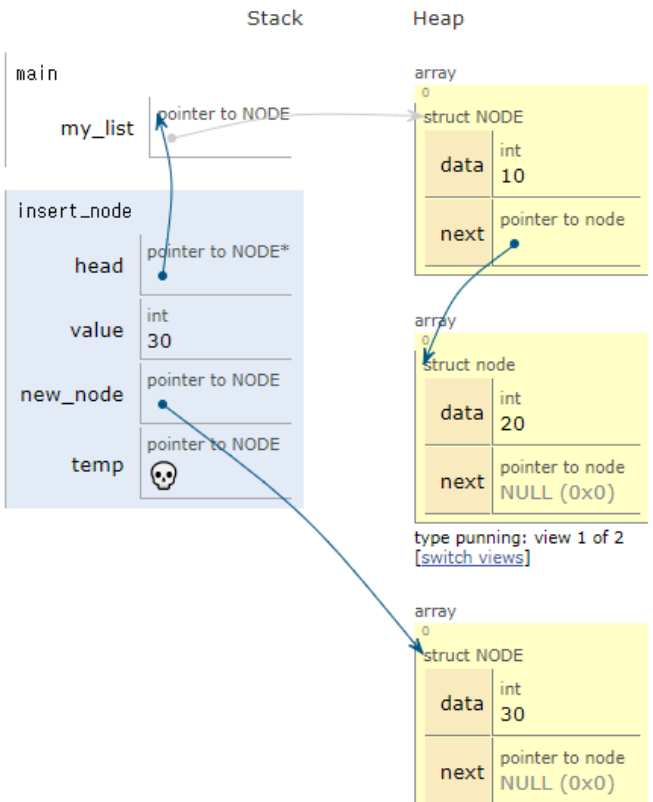
```
31    NODE *new_node = create_node(value);
32
33    if(*head == NULL)
34      {
35        // 리스트가 비어있을 경우
36        *head = new_node;
37      }
38    else
39      {
40        // 리스트가 비어있지 않을 경우
41        NODE *temp = *head;
42        while(temp->next != NULL)
43          {
44            temp = temp->next;
45          }
46        temp->next = new_node;
47      }
48  }
49
50  // 연결 리스트의 노드를 출력하는 함수
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



그림으로 설명

C (C17 + GNU extensions)
known limitations

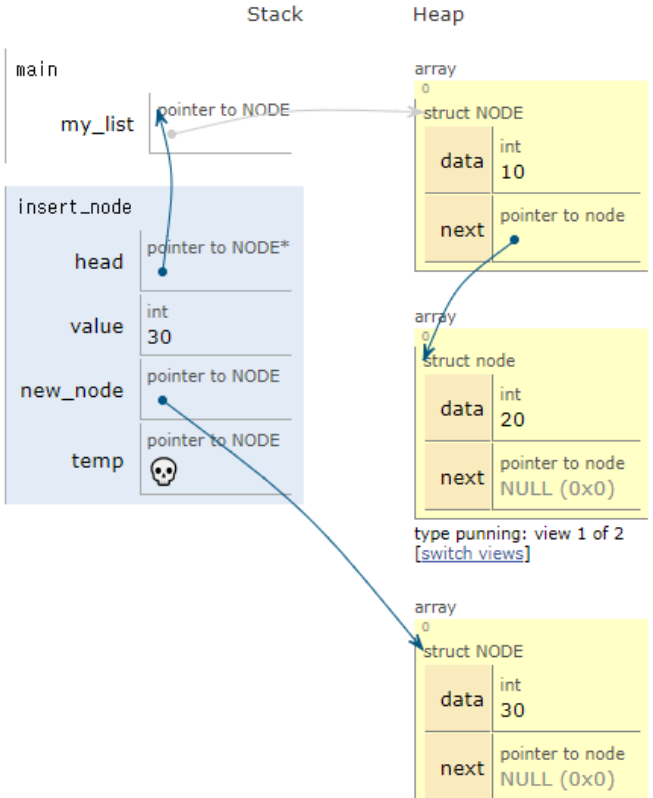
```
31    NODE *new_node = create_node(value);
32
33    if(*head == NULL)
34      {
35        // 리스트가 비어있을 경우
36        *head = new_node;
37      }
38    else
39      {
40        // 리스트가 비어있지 않을 경우
41        NODE *temp = *head;
42        while(temp->next != NULL)
43          {
44            temp = temp->next;
45          }
46        temp->next = new_node;
47      }
48  }
49
50  // 연결 리스트의 노드를 출력하는 함수
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



그림으로 설명

C (C17 + GNU extensions)
known limitations

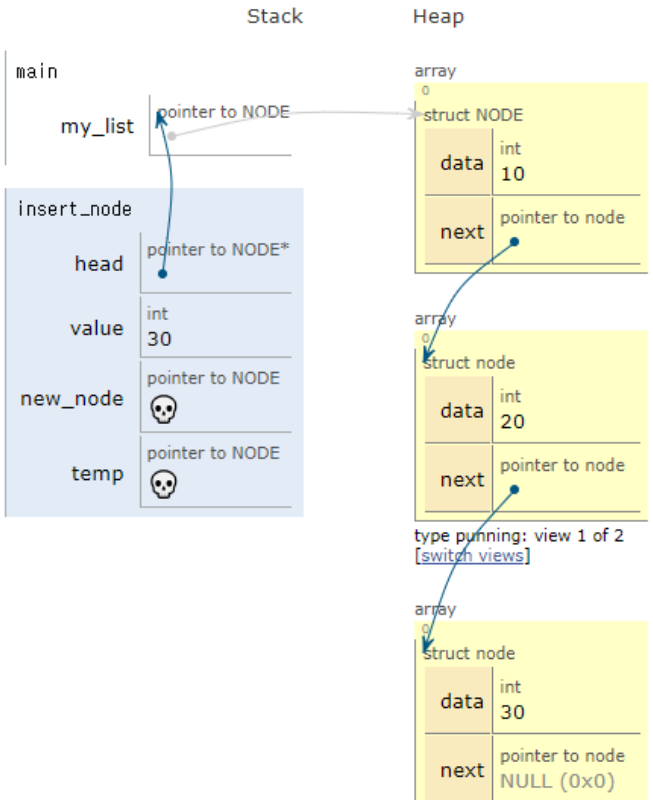
```
{
  NODE *new_node = create_node(value);

  if(*head == NULL)
    {
      // 리스트가 비어있을 경우
      *head = new_node;
    }
  else
    {
      // 리스트가 비어있지 않을 경우
      NODE *temp = *head;
      while(temp->next != NULL)
        {
          temp = temp->next;
        }
      temp->next = new_node;
    }
}

// 연결 리스트의 노드를 출력하는 함수
void print_list(NODE *head)
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



그림으로 설명

C (C17 + GNU extensions)
known limitations

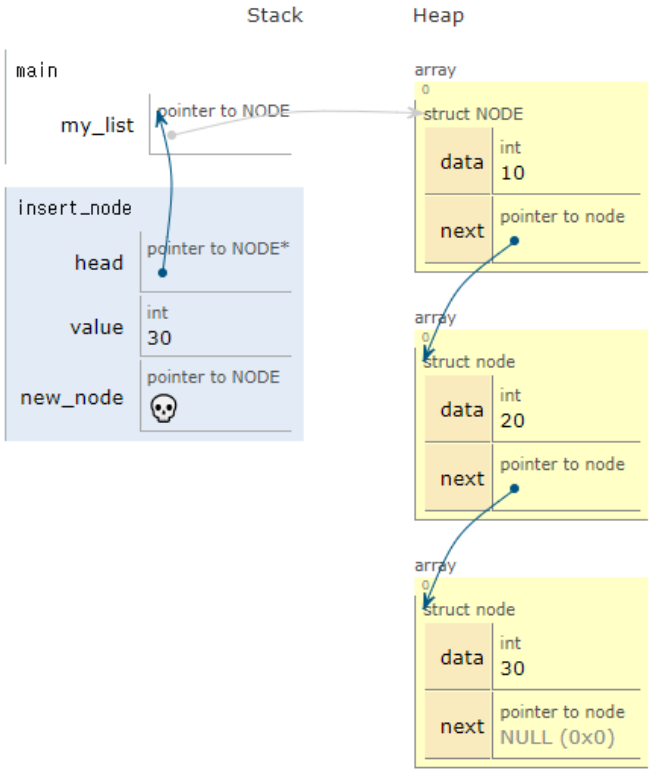
```
31    NODE *new_node = create_node(value);
32
33    if(*head == NULL)
34      {
35        // 리스트가 비어있을 경우
36        *head = new_node;
37      }
38    else
39      {
40        // 리스트가 비어있지 않을 경우
41        NODE *temp = *head;
42        while(temp->next != NULL)
43          {
44            temp = temp->next;
45          }
46        temp->next = new_node;
47      }
48  }
49
50  // 연결 리스트의 노드를 출력하는 함수
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



type punning: view 1 of 2

그림으로 설명

C (C17 + GNU extensions)
known limitations

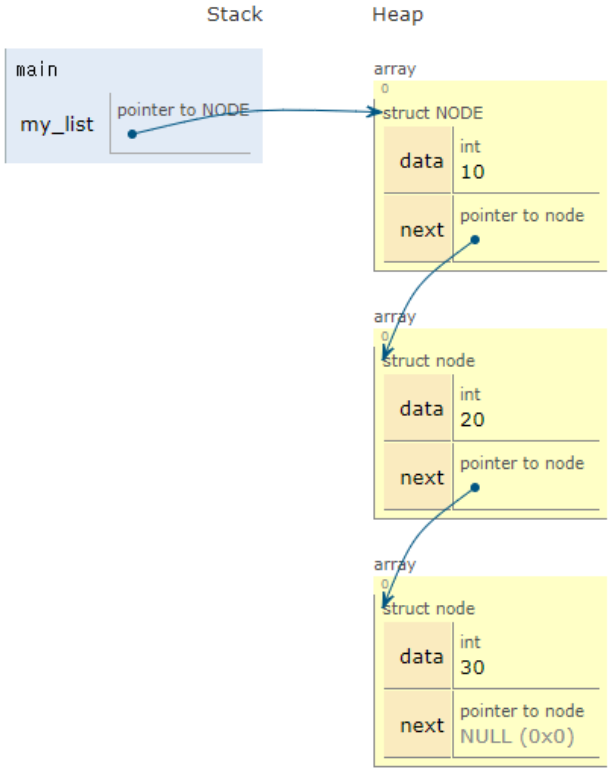
```
  printf("메모리 해제 완료\n");
}

int main()
{
  NODE *my_list = NULL;

  // 연결 리스트에 노드 추가
  insert_node(&my_list, 10);
  insert_node(&my_list, 20);
  insert_node(&my_list, 30);

  // 연결 리스트 출력
  printf("연결 리스트: ");
  print_list(my_list);

  // 메모리 해제
  free_list(my_list);

  return 0;
}
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)



그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  printf("메모리 해제 완료\n");
}

int main()
{
  NODE *my_list = NULL;

  // 연결 리스트에 노드 추가
  insert_node(&my_list, 10);
  insert_node(&my_list, 20);
  insert_node(&my_list, 30);

  // 연결 리스트 출력
  printf("연결 리스트: ");
  print_list(my_list);

  // 메모리 해제
  free_list(my_list);

  return 0;
}
```

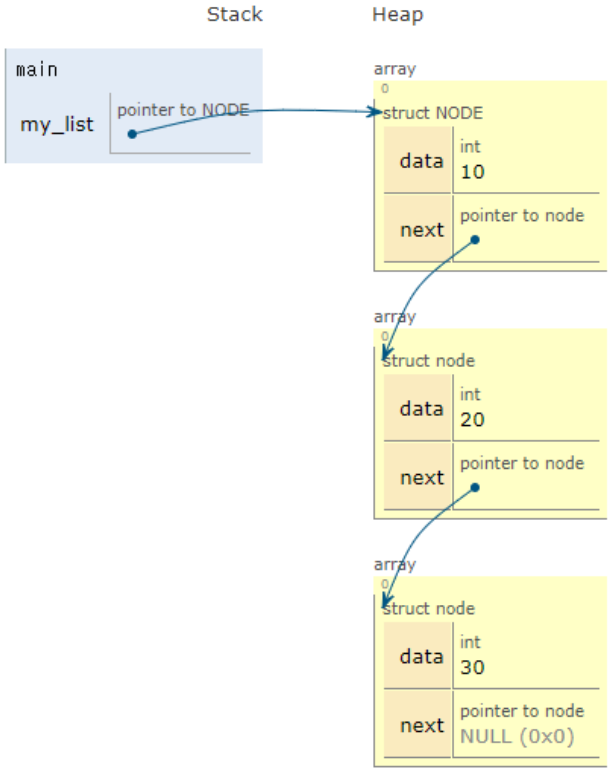
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트:
```



그림으로 설명

C (C17 + GNU extensions)
known limitations

```
42        while(temp->next != NULL)
43          {
44            temp = temp->next;
45          }
46        temp->next = new_node;
47      }
48  }
49
50  // 연결 리스트의 노드를 출력하는 함수
51  void print_list(NODE *head)
52  {
53    NODE *temp = head;
54    while(temp != NULL)
55      {
56        printf("%d -> ", temp->data);
57        temp = temp->next;
58      }
59    printf("NULL₩n");
60  }
61
```

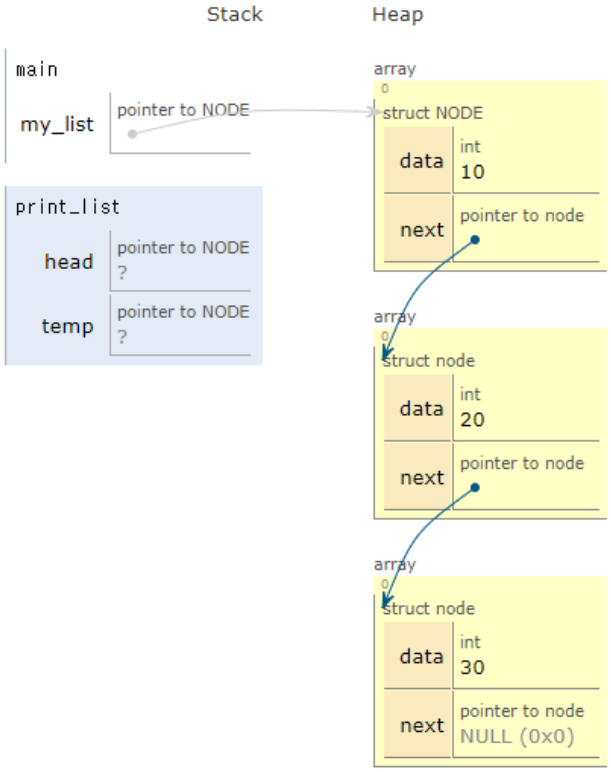
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트:
```



그림으로 설명

C (C17 + GNU extensions)
known limitations

```
42        while(temp->next != NULL)
43          {
44            temp = temp->next;
45          }
46        temp->next = new_node;
47      }
48  }
49
50  // 연결 리스트의 노드를 출력하는 함수
51  void print_list(NODE *head)
52  {
53    NODE *temp = head;
54    while(temp != NULL)
55      {
56        printf("%d -> ", temp->data);
57        temp = temp->next;
58      }
59    printf("NULL₩n");
60  }
61
```

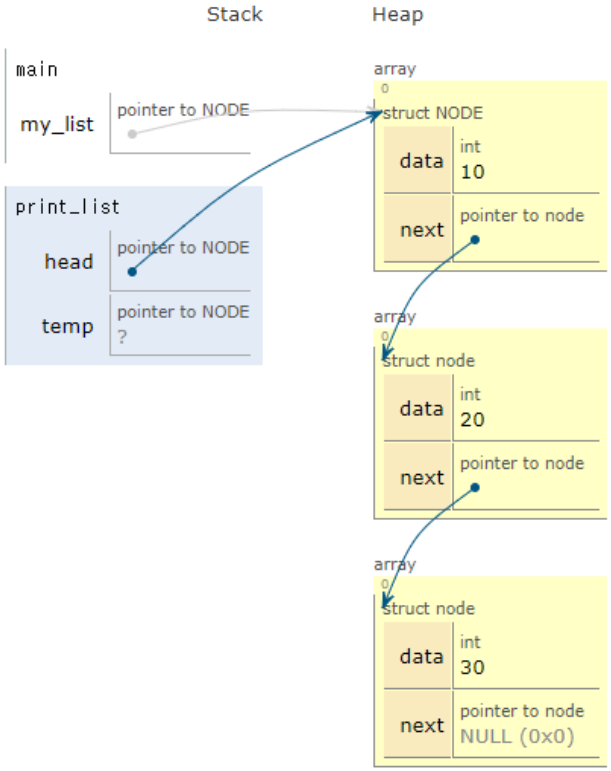
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트:
```



그림으로 설명

C (C17 + GNU extensions)
known limitations

```
42        while(temp->next != NULL)
43          {
44            temp = temp->next;
45          }
46        temp->next = new_node;
47      }
48  }
49
50  // 연결 리스트의 노드를 출력하는 함수
51  void print_list(NODE *head)
52  {
53    NODE *temp = head;
54    while(temp != NULL)
55      {
56        printf("%d -> ", temp->data);
57        temp = temp->next;
58      }
59    printf("NULL₩n");
60  }
61
```

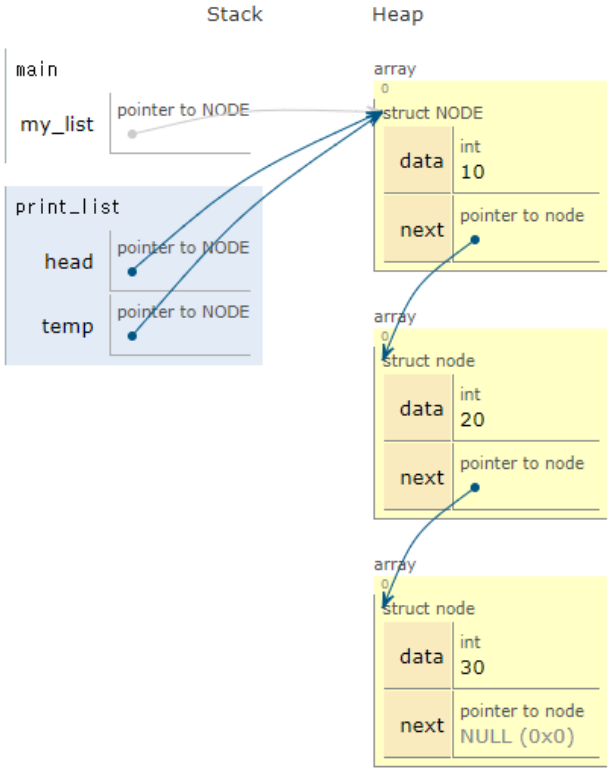
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트:
```



그림으로 설명

C (C17 + GNU extensions)
known limitations

```
42        while(temp->next != NULL)
43          {
44            temp = temp->next;
45          }
46        temp->next = new_node;
47      }
48  }
49
50  // 연결 리스트의 노드를 출력하는 함수
51  void print_list(NODE *head)
52  {
53    NODE *temp = head;
54    while(temp != NULL)
55      {
56        printf("%d -> ", temp->data);
57        temp = temp->next;
58      }
59    printf("NULL₩n");
60  }
61
```

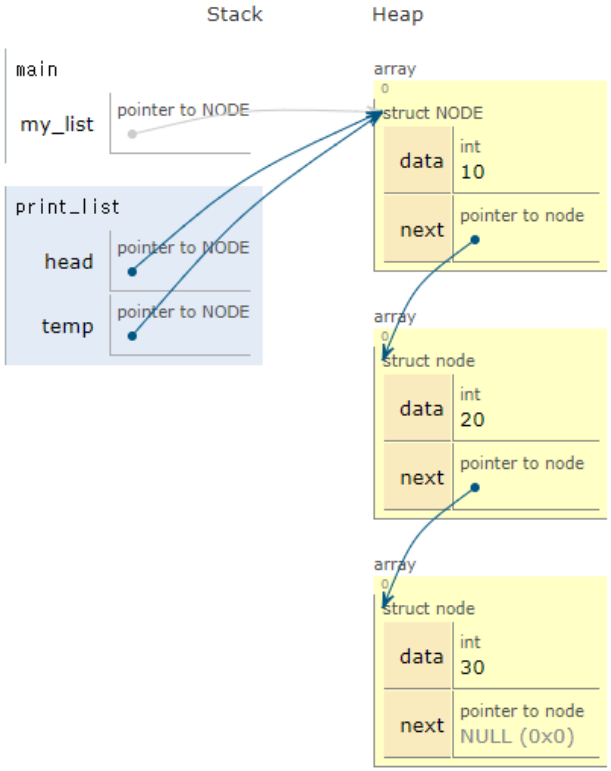
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트:
```



그림으로 설명

C (C17 + GNU extensions)
known limitations

```
42        while(temp->next != NULL)
43          {
44            temp = temp->next;
45          }
46        temp->next = new_node;
47      }
48  }
49
50  // 연결 리스트의 노드를 출력하는 함수
51  void print_list(NODE *head)
52  {
53    NODE *temp = head;
54    while(temp != NULL)
55      {
56        printf("%d -> ", temp->data);
57        temp = temp->next;
58      }
59    printf("NULL₩n");
60  }
61
```

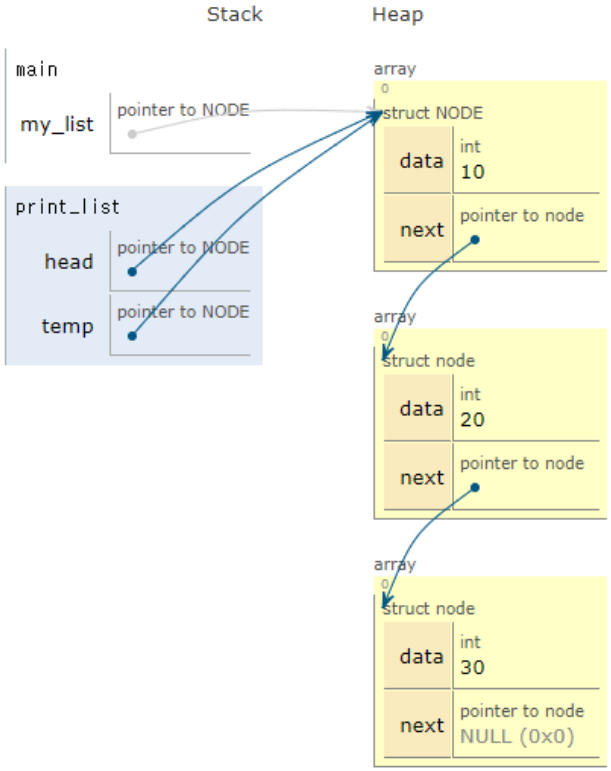
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 ->
```



그림으로 설명

C (C17 + GNU extensions)
known limitations

```
42        while(temp->next != NULL)
43          {
44            temp = temp->next;
45          }
46        temp->next = new_node;
47      }
48  }
49
50  // 연결 리스트의 노드를 출력하는 함수
51  void print_list(NODE *head)
52  {
53    NODE *temp = head;
54    while(temp != NULL)
55      {
56        printf("%d -> ", temp->data);
57        temp = temp->next;
58      }
59    printf("NULL₩n");
60  }
61
```

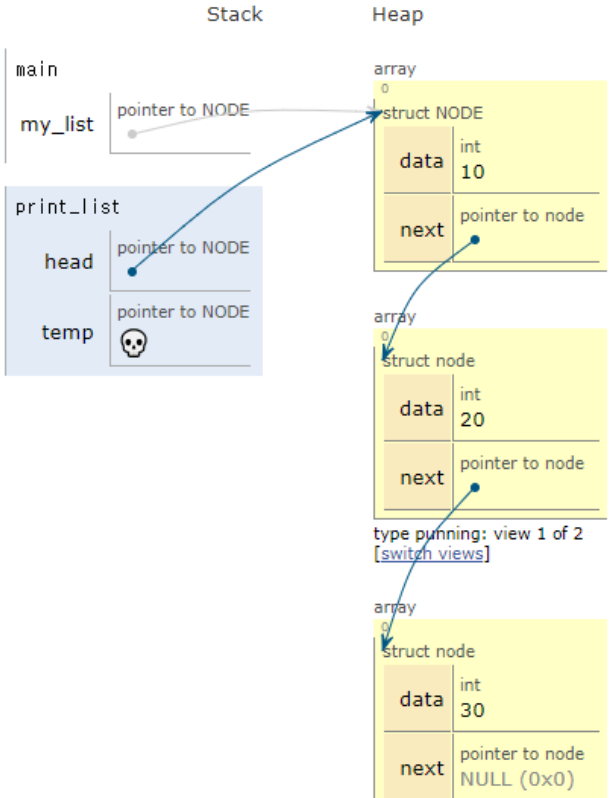
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 ->
```



그림으로 설명

C (C17 + GNU extensions)
known limitations

```
42        while(temp->next != NULL)
43          {
44            temp = temp->next;
45          }
46        temp->next = new_node;
47      }
48  }
49
50  // 연결 리스트의 노드를 출력하는 함수
51  void print_list(NODE *head)
52  {
53    NODE *temp = head;
54    while(temp != NULL)
55      {
56        printf("%d -> ", temp->data);
57        temp = temp->next;
58      }
59    printf("NULL₩n");
60  }
61
```

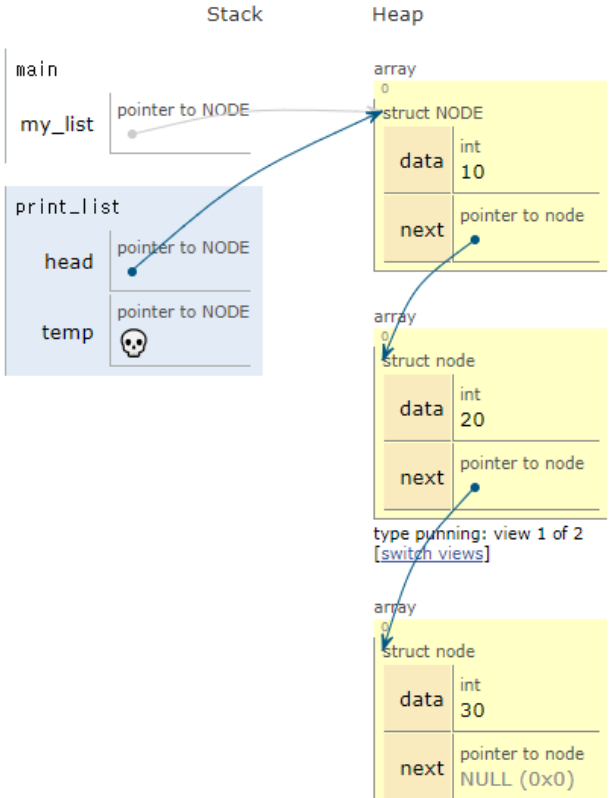
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 ->
```



그림으로 설명

C (C17 + GNU extensions)
known limitations

```
42        while(temp->next != NULL)
43          {
44            temp = temp->next;
45          }
46        temp->next = new_node;
47      }
48  }
49
50  // 연결 리스트의 노드를 출력하는 함수
51  void print_list(NODE *head)
52  {
53    NODE *temp = head;
54    while(temp != NULL)
55      {
56        printf("%d -> ", temp->data);
57        temp = temp->next;
58      }
59    printf("NULL₩n");
60  }
61
```

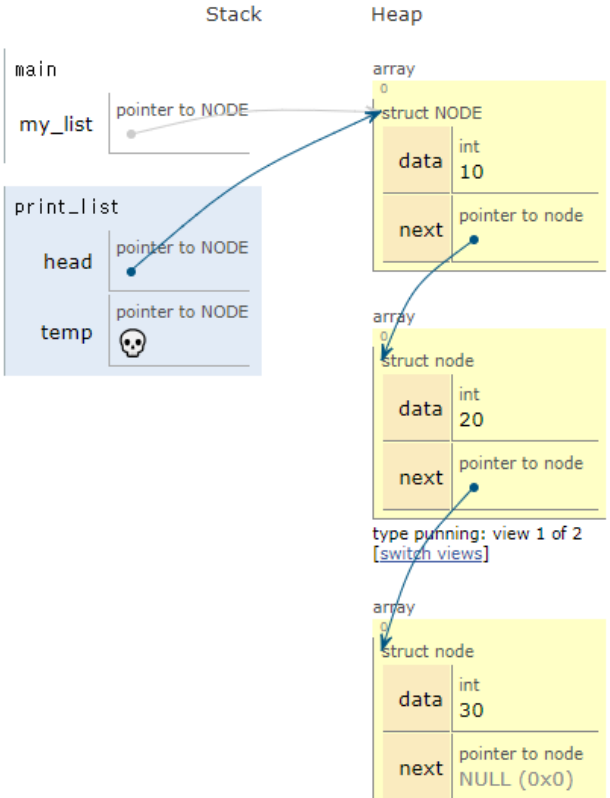
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 ->
```



그림으로 설명

C (C17 + GNU extensions)
known limitations

```
42        while(temp->next != NULL)
43          {
44            temp = temp->next;
45          }
46        temp->next = new_node;
47      }
48  }
49
50  // 연결 리스트의 노드를 출력하는 함수
51  void print_list(NODE *head)
52  {
53    NODE *temp = head;
54    while(temp != NULL)
55      {
56        printf("%d -> ", temp->data);
57        temp = temp->next;
58      }
59    printf("NULL₩n");
60  }
61
```

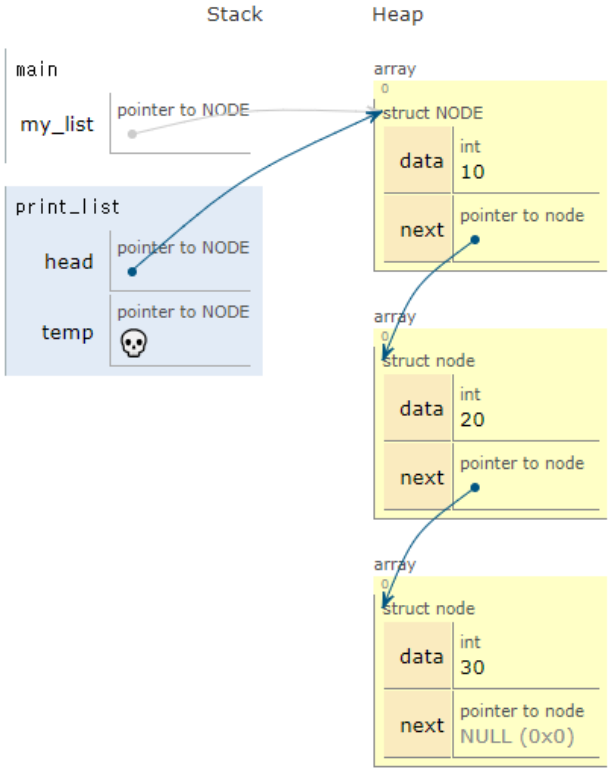
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 ->
```



type punning: view 1 of 2

그림으로 설명

C (C17 + GNU extensions)
known limitations

```
42        while(temp->next != NULL)
43          {
44            temp = temp->next;
45          }
46        temp->next = new_node;
47      }
48  }
49
50  // 연결 리스트의 노드를 출력하는 함수
51  void print_list(NODE *head)
52  {
53    NODE *temp = head;
54    while(temp != NULL)
55      {
56        printf("%d -> ", temp->data);
57        temp = temp->next;
58      }
59    printf("NULL₩n");
60  }
61
```

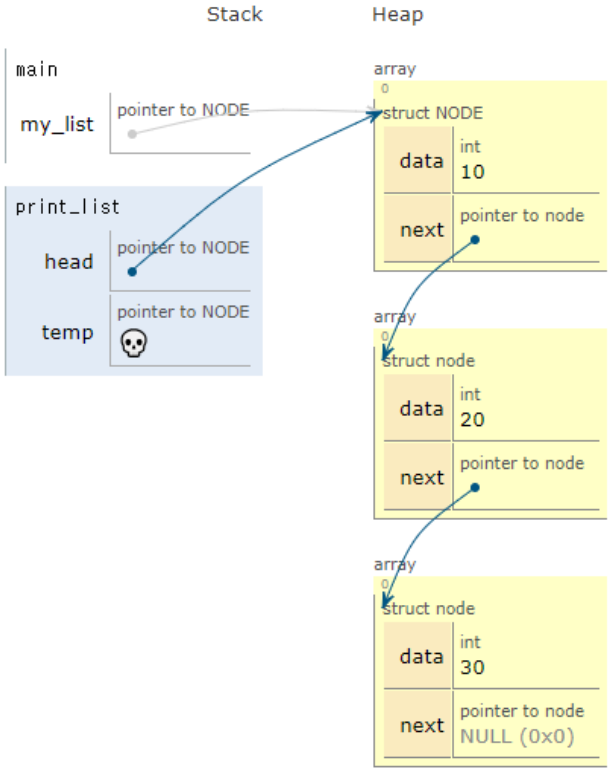
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 ->
```



type punning: view 1 of 2

그림으로 설명

C (C17 + GNU extensions)
known limitations

```
42        while(temp->next != NULL)
43          {
44            temp = temp->next;
45          }
46        temp->next = new_node;
47      }
48  }
49
50  // 연결 리스트의 노드를 출력하는 함수
51  void print_list(NODE *head)
52  {
53    NODE *temp = head;
54    while(temp != NULL)
55      {
56        printf("%d -> ", temp->data);
57        temp = temp->next;
58      }
59    printf("NULL₩n");
60  }
61
```

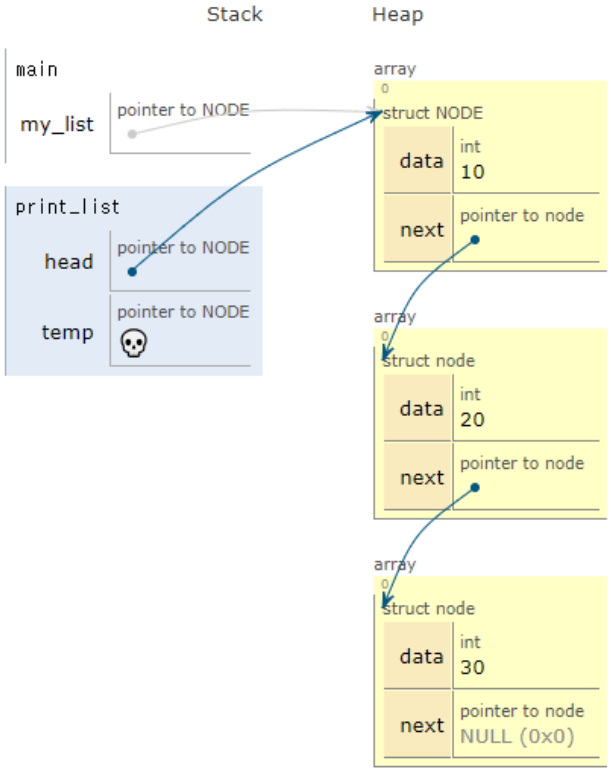
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 ->
```



type punning: view 1 of 2

그림으로 설명

C (C17 + GNU extensions)
known limitations

```
42        while(temp->next != NULL)
43          {
44            temp = temp->next;
45          }
46        temp->next = new_node;
47      }
48  }
49
50  // 연결 리스트의 노드를 출력하는 함수
51  void print_list(NODE *head)
52  {
53    NODE *temp = head;
54    while(temp != NULL)
55      {
56        printf("%d -> ", temp->data);
57        temp = temp->next;
58      }
59    printf("NULL₩n");
60  }
61
```

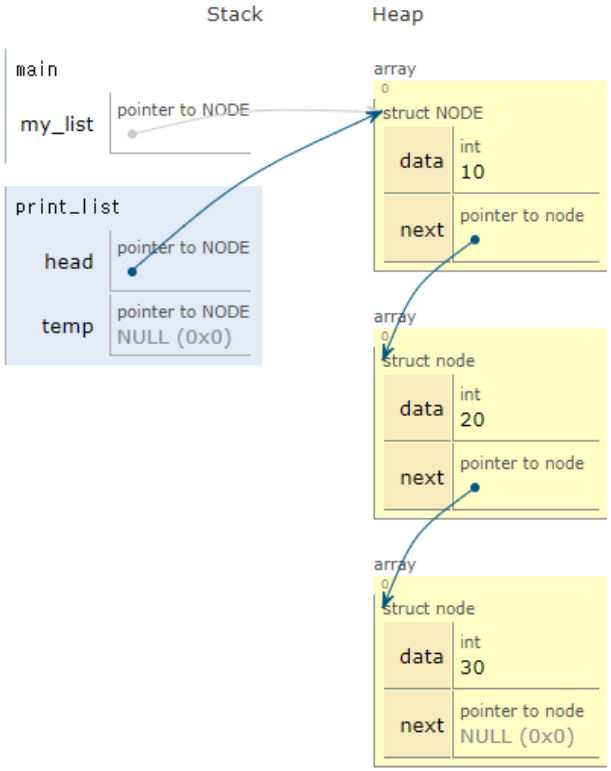
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 ->
```



그림으로 설명

C (C17 + GNU extensions)
known limitations

```
42        while(temp->next != NULL)
43          {
44            temp = temp->next;
45          }
46        temp->next = new_node;
47      }
48  }
49
50  // 연결 리스트의 노드를 출력하는 함수
51  void print_list(NODE *head)
52  {
53    NODE *temp = head;
54    while(temp != NULL)
55      {
56        printf("%d -> ", temp->data);
57        temp = temp->next;
58      }
59    printf("NULL₩n");
60  }
61
```

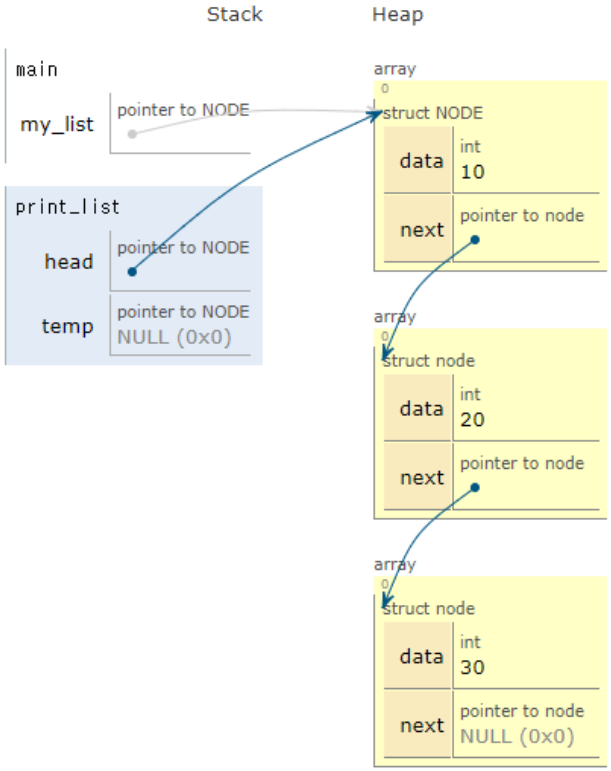
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 ->
```



그림으로 설명

C (C17 + GNU extensions)
known limitations

```
42        while(temp->next != NULL)
43          {
44            temp = temp->next;
45          }
46        temp->next = new_node;
47      }
48  }
49
50  // 연결 리스트의 노드를 출력하는 함수
51  void print_list(NODE *head)
52  {
53    NODE *temp = head;
54    while(temp != NULL)
55      {
56        printf("%d -> ", temp->data);
57        temp = temp->next;
58      }
59    printf("NULL₩n");
60  }
61
```

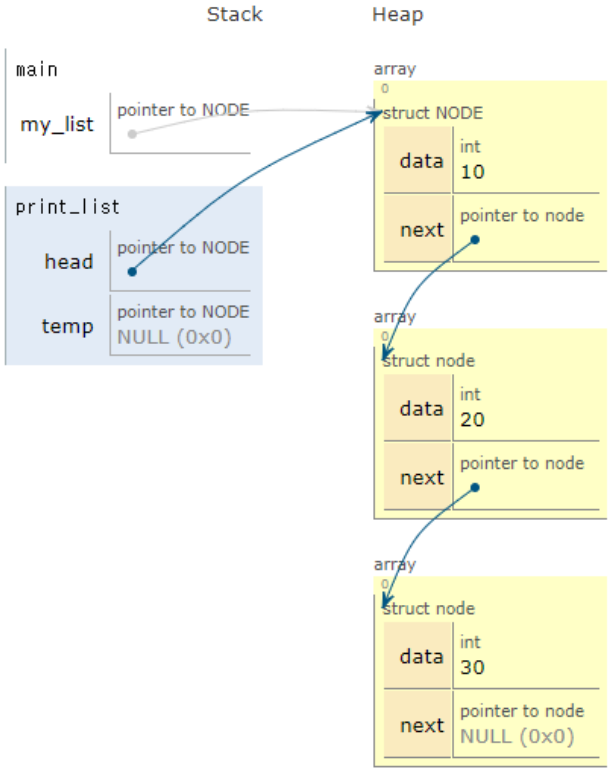
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
```



그림으로 설명

C (C17 + GNU extensions)
known limitations

```
42        while(temp->next != NULL)
43          {
44            temp = temp->next;
45          }
46        temp->next = new_node;
47      }
48  }
49
50  // 연결 리스트의 노드를 출력하는 함수
51  void print_list(NODE *head)
52  {
53    NODE *temp = head;
54    while(temp != NULL)
55      {
56        printf("%d -> ", temp->data);
57        temp = temp->next;
58      }
59    printf("NULL₩n");
60  }
61
```

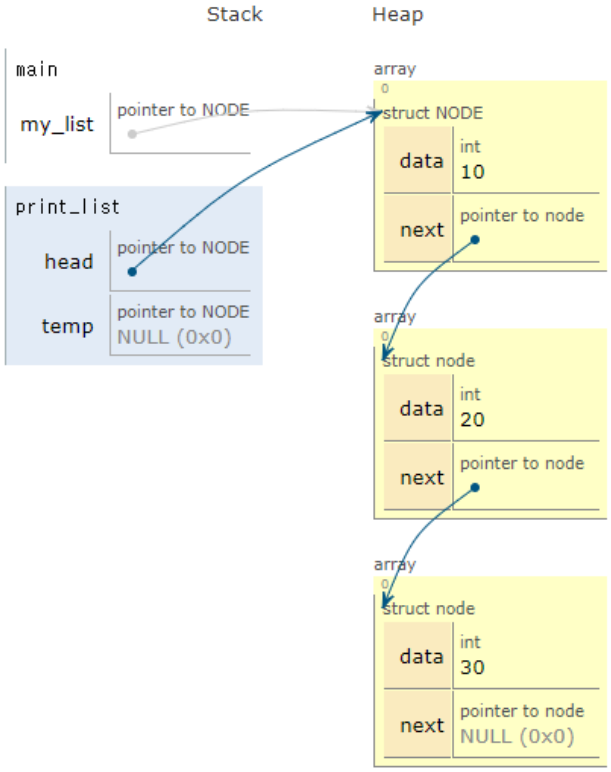
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
```



그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  printf("메모리 해제 완료\n");
}

int main()
{
  NODE *my_list = NULL;

  // 연결 리스트에 노드 추가
  insert_node(&my_list, 10);
  insert_node(&my_list, 20);
  insert_node(&my_list, 30);

  // 연결 리스트 출력
  printf("연결 리스트: ");
  print_list(my_list);

  // 메모리 해제
  free_list(my_list);

  return 0;
}
```

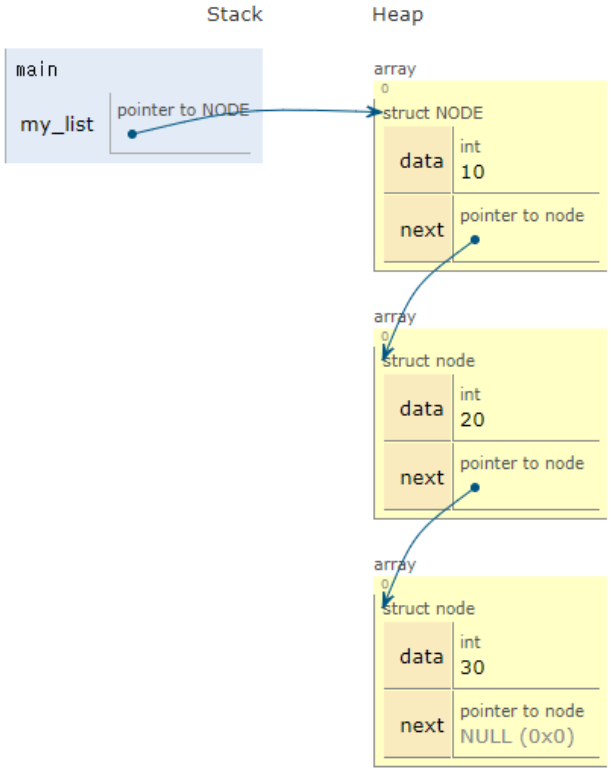
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
```



그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  while(temp != NULL)
    {
      printf("%d -> ", temp->data);
      temp = temp->next;
    }
  printf("NULL₩n");
}

// 메모리 해제 함수
void free_list(NODE *head)
{
  NODE *temp;
  while(head != NULL)
    {
      temp = head;
      head = head->next;
      free(temp);
    }
  printf("메모리 해제 완료₩n");
}
```

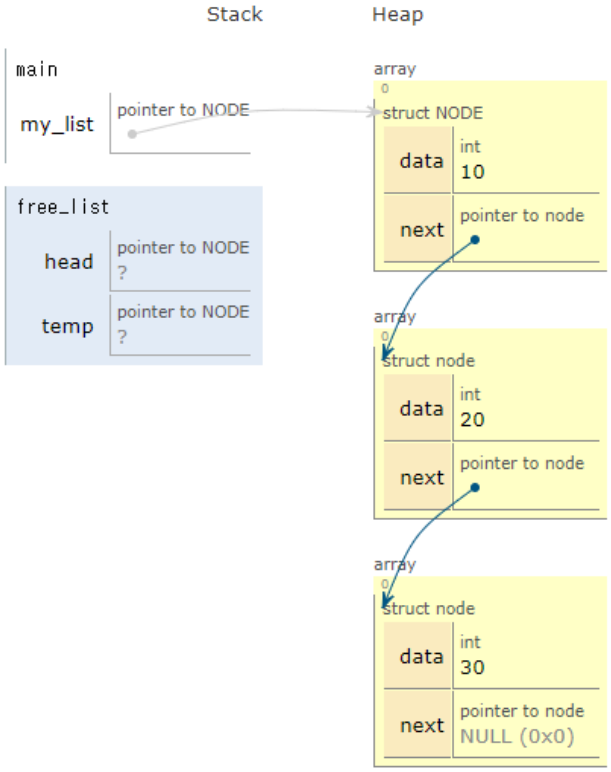
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
```



그림으로 설명

C (C17 + GNU extensions)
known limitations

```
54   while(temp != NULL)
55     {
56       printf("%d -> ", temp->data);
57       temp = temp->next;
58     }
59   printf("NULL₩n");
60 }
61
62 // 메모리 해제 함수
63 void free_list(NODE *head)
64 {
65   NODE *temp;
66   while(head != NULL)
67     {
68       temp = head;
69       head = head->next;
70       free(temp);
71     }
72   printf("메모리 해제 완료₩n");
73 }
```

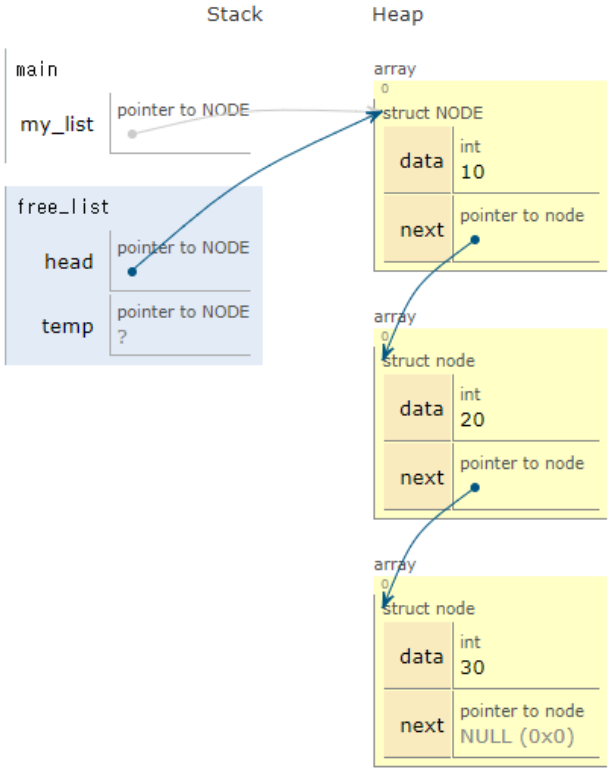
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
```



그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  while(temp != NULL)
    {
      printf("%d -> ", temp->data);
      temp = temp->next;
    }
  printf("NULL₩n");
}

// 메모리 해제 함수
void free_list(NODE *head)
{
  NODE *temp;
  while(head != NULL)
    {
      temp = head;
      head = head->next;
      free(temp);
    }
  printf("메모리 해제 완료₩n");
}
```

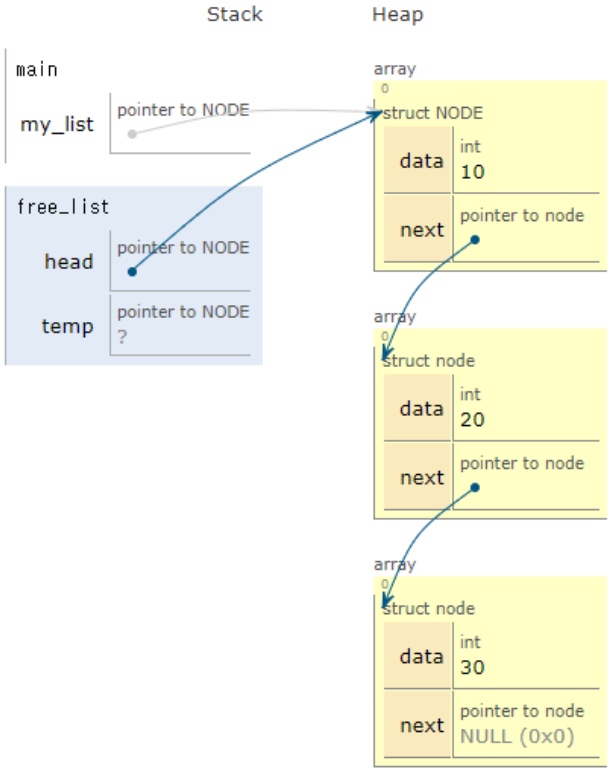
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
```



그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  while(temp != NULL)
    {
      printf("%d -> ", temp->data);
      temp = temp->next;
    }
  printf("NULL₩n");
}

// 메모리 해제 함수
void free_list(NODE *head)
{
  NODE *temp;
  while(head != NULL)
    {
      temp = head;
      head = head->next;
      free(temp);
    }
  printf("메모리 해제 완료₩n");
}
```

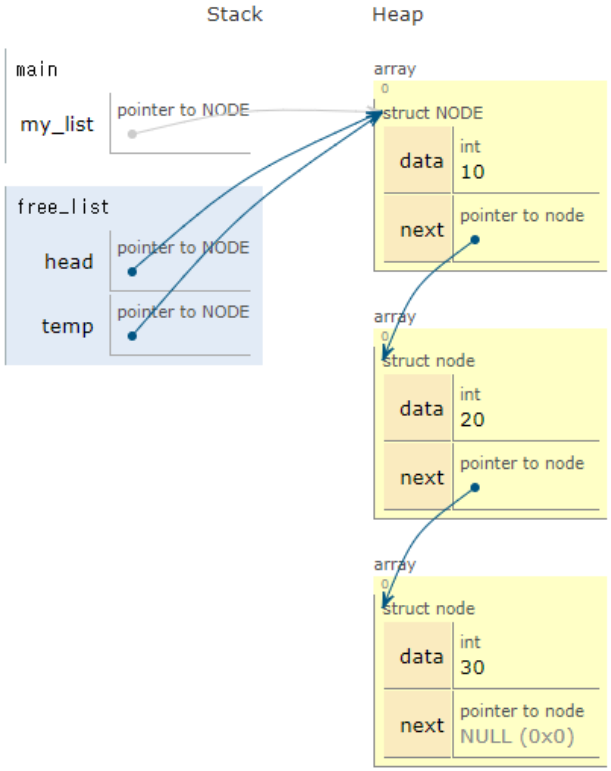
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
```



그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  while(temp != NULL)
    {
      printf("%d -> ", temp->data);
      temp = temp->next;
    }
  printf("NULL₩n");
}

// 메모리 해제 함수
void free_list(NODE *head)
{
  NODE *temp;
  while(head != NULL)
    {
      temp = head;
      head = head->next;
      free(temp);
    }
  printf("메모리 해제 완료₩n");
}
```

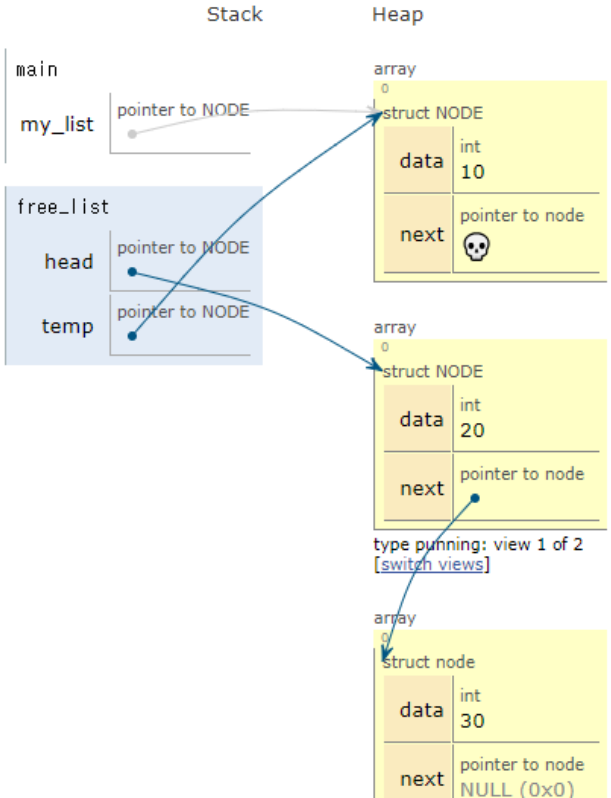
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
```



그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  while(temp != NULL)
    {
      printf("%d -> ", temp->data);
      temp = temp->next;
    }
  printf("NULL₩n");
}

// 메모리 해제 함수
void free_list(NODE *head)
{
  NODE *temp;
  while(head != NULL)
    {
      temp = head;
      head = head->next;
      free(temp);
    }
  printf("메모리 해제 완료₩n");
}
```

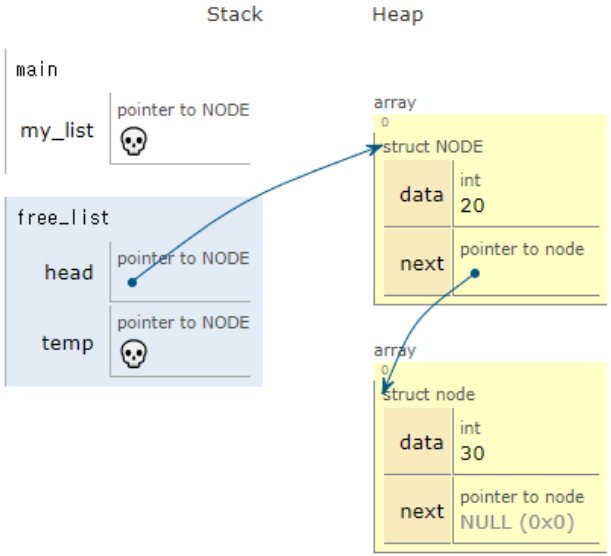
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
```



Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

Note: 💀 means a pointer points to memory that is either unallocated or misaligned with data boundaries. 💀 locations are *approximate* and may not match the pointer's real address.

그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  while(temp != NULL)
    {
      printf("%d -> ", temp->data);
      temp = temp->next;
    }
  printf("NULL₩n");
}

// 메모리 해제 함수
void free_list(NODE *head)
{
  NODE *temp;
  while(head != NULL)
    {
      temp = head;
      head = head->next;
      free(temp);
    }
  printf("메모리 해제 완료₩n");
}
```

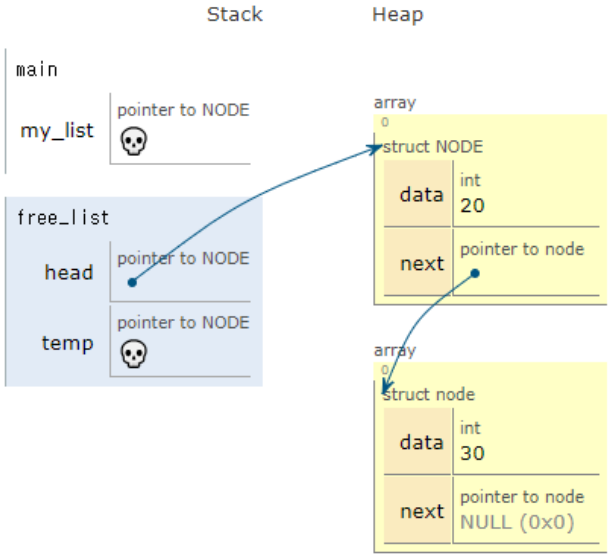
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
```



Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

Note: 💀 means a pointer points to memory that is either unallocated or misaligned with data boundaries. 💀 locations are *approximate* and may not match the pointer's real address.

그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  while(temp != NULL)
    {
      printf("%d -> ", temp->data);
      temp = temp->next;
    }
  printf("NULL₩n");
}

// 메모리 해제 함수
void free_list(NODE *head)
{
  NODE *temp;
  while(head != NULL)
    {
      temp = head;
      head = head->next;
      free(temp);
    }
  printf("메모리 해제 완료₩n");
}
```

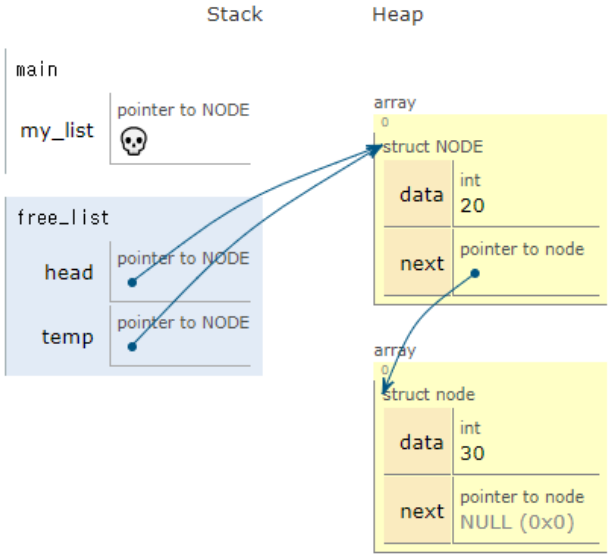
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
```



Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

Note: 💀 means a pointer points to memory that is either unallocated or misaligned with data boundaries. 💀 locations are *approximate* and may not match the pointer's real address.

그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  while(temp != NULL)
    {
      printf("%d -> ", temp->data);
      temp = temp->next;
    }
  printf("NULL₩n");
}

// 메모리 해제 함수
void free_list(NODE *head)
{
  NODE *temp;
  while(head != NULL)
    {
      temp = head;
      head = head->next;
      free(temp);
    }
  printf("메모리 해제 완료₩n");
}
```

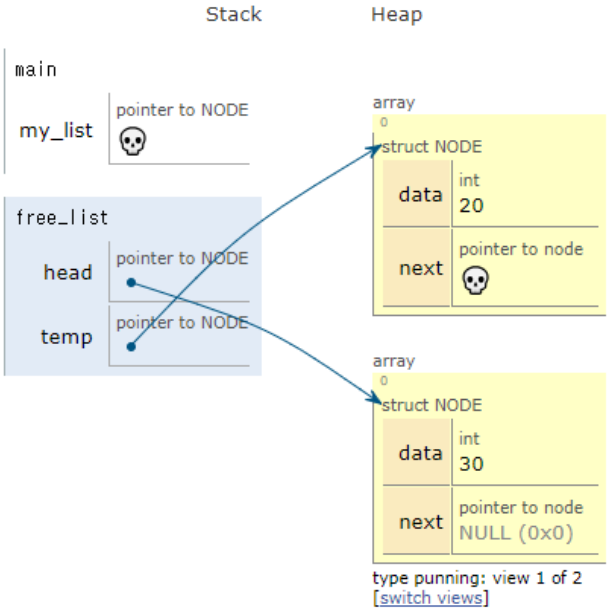
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
```



type punning: view 1 of 2
[switch views]

Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

Note: 💀 means a pointer points to memory that is either

그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  while(temp != NULL)
    {
      printf("%d -> ", temp->data);
      temp = temp->next;
    }
  printf("NULL₩n");
}

// 메모리 해제 함수
void free_list(NODE *head)
{
  NODE *temp;
  while(head != NULL)
    {
      temp = head;
      head = head->next;
      free(temp);
    }
  printf("메모리 해제 완료₩n");
}
```

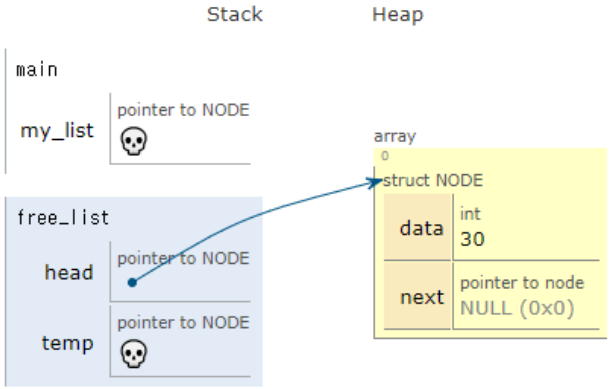
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
```



Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

Note: 💀 means a pointer points to memory that is either unallocated or misaligned with data boundaries. 💀 locations are *approximate* and may not match the pointer's real address. Select 'byte-level view of data' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  while(temp != NULL)
    {
      printf("%d -> ", temp->data);
      temp = temp->next;
    }
  printf("NULL₩n");
}

// 메모리 해제 함수
void free_list(NODE *head)
{
  NODE *temp;
  while(head != NULL)
    {
      temp = head;
      head = head->next;
      free(temp);
    }
  printf("메모리 해제 완료₩n");
}
```

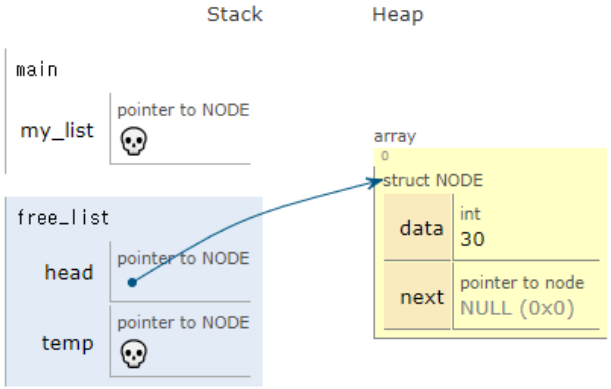
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
```



Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

Note: 💀 means a pointer points to memory that is either unallocated or misaligned with data boundaries. 💀 locations are *approximate* and may not match the pointer's real address. Select 'byte-level view of data' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  while(temp != NULL)
    {
      printf("%d -> ", temp->data);
      temp = temp->next;
    }
  printf("NULL₩n");
}

// 메모리 해제 함수
void free_list(NODE *head)
{
  NODE *temp;
  while(head != NULL)
    {
      temp = head;
      head = head->next;
      free(temp);
    }
  printf("메모리 해제 완료₩n");
}
```

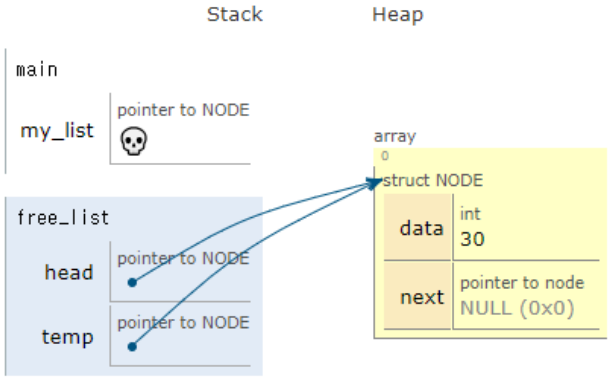
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
```



Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

Note: 💀 means a pointer points to memory that is either unallocated or misaligned with data boundaries. 💀 locations are *approximate* and may not match the pointer's real address. Select 'byte-level view of data' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  while(temp != NULL)
    {
      printf("%d -> ", temp->data);
      temp = temp->next;
    }
  printf("NULL₩n");
}

// 메모리 해제 함수
void free_list(NODE *head)
{
  NODE *temp;
  while(head != NULL)
    {
      temp = head;
      head = head->next;
      free(temp);
    }
  printf("메모리 해제 완료₩n");
}
```

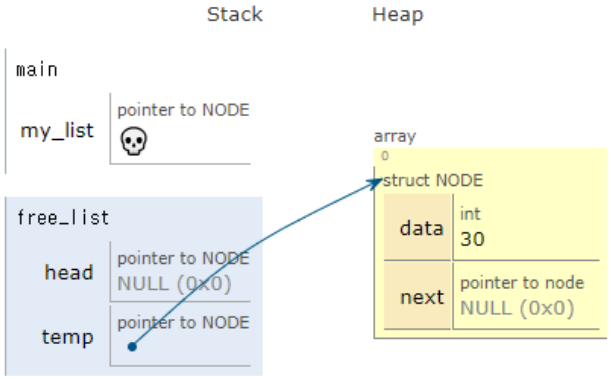
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
```



Note: Pointers to structs, unions, and C++ objects may be ambiguous since a struct/object's address is the same as its first member and a union's address is the same as all its members. Select 'show memory addresses' below to see more details:

Note: 💀 means a pointer points to memory that is either unallocated or misaligned with data boundaries. 💀 locations are *approximate* and may not match the pointer's real address. Select 'byte-level view of data' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  while(temp != NULL)
    {
      printf("%d -> ", temp->data);
      temp = temp->next;
    }
  printf("NULL₩n");
}

// 메모리 해제 함수
void free_list(NODE *head)
{
  NODE *temp;
  while(head != NULL)
    {
      temp = head;
      head = head->next;
      free(temp);
    }
  printf("메모리 해제 완료₩n");
}
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
```

Stack | Heap

main

my_list	pointer to NODE 💀

free_list

head	pointer to NODE NULL (0x0)
temp	pointer to NODE 💀

Note: 💀 means a pointer points to memory that is either unallocated or misaligned with data boundaries. 💀 locations are *approximate* and may not match the pointer's real address. Select 'byte-level view of data' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  while(temp != NULL)
    {
      printf("%d -> ", temp->data);
      temp = temp->next;
    }
  printf("NULL₩n");
}

// 메모리 해제 함수
void free_list(NODE *head)
{
  NODE *temp;
  while(head != NULL)
    {
      temp = head;
      head = head->next;
      free(temp);
    }
  printf("메모리 해제 완료₩n");
}
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
```

Stack | Heap

main

my_list	pointer to NODE 💀

free_list

head	pointer to NODE NULL (0x0)
temp	pointer to NODE 💀

Note: 💀 means a pointer points to memory that is either unallocated or misaligned with data boundaries. 💀 locations are *approximate* and may not match the pointer's real address. Select 'byte-level view of data' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  while(temp != NULL)
    {
      printf("%d -> ", temp->data);
      temp = temp->next;
    }
  printf("NULL₩n");
}

// 메모리 해제 함수
void free_list(NODE *head)
{
  NODE *temp;
  while(head != NULL)
    {
      temp = head;
      head = head->next;
      free(temp);
    }
  printf("메모리 해제 완료₩n");
}
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
메모리 해제 완료
```

Stack Heap

main

my_list	pointer to NODE 💀

free_list

head	pointer to NODE NULL (0x0)
temp	pointer to NODE 💀

Note: 💀 means a pointer points to memory that is either unallocated or misaligned with data boundaries. 💀 locations are *approximate* and may not match the pointer's real address. Select 'byte-level view of data' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  while(temp != NULL)
    {
      printf("%d -> ", temp->data);
      temp = temp->next;
    }
  printf("NULL₩n");
}

// 메모리 해제 함수
void free_list(NODE *head)
{
  NODE *temp;
  while(head != NULL)
    {
      temp = head;
      head = head->next;
      free(temp);
    }
  printf("메모리 해제 완료₩n");
}
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
메모리 해제 완료
```

Stack | Heap

main

my_list	pointer to NODE 💀

free_list

head	pointer to NODE NULL (0x0)
temp	pointer to NODE 💀

Note: 💀 means a pointer points to memory that is either unallocated or misaligned with data boundaries. 💀 locations are *approximate* and may not match the pointer's real address. Select 'byte-level view of data' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  printf("메모리 해제 완료\n");
}

int main()
{
  NODE *my_list = NULL;

  // 연결 리스트에 노드 추가
  insert_node(&my_list, 10);
  insert_node(&my_list, 20);
  insert_node(&my_list, 30);

  // 연결 리스트 출력
  printf("연결 리스트: ");
  print_list(my_list);

  // 메모리 해제
  free_list(my_list);

  return 0;
}
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
메모리 해제 완료
```

Stack Heap

main

my_list | pointer to NODE 💀

Note: 💀 means a pointer points to memory that is either unallocated or misaligned with data boundaries. 💀 locations are *approximate* and may not match the pointer's real address. Select 'byte-level view of data' below to see more details:

C/C++ details: none [default view]

그림으로 설명

C (C17 + GNU extensions)
known limitations

```
  printf("메모리 해제 완료\n");
}

int main()
{
  NODE *my_list = NULL;

  // 연결 리스트에 노드 추가
  insert_node(&my_list, 10);
  insert_node(&my_list, 20);
  insert_node(&my_list, 30);

  // 연결 리스트 출력
  printf("연결 리스트: ");
  print_list(my_list);

  // 메모리 해제
  free_list(my_list);

  return 0;
}
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
연결 리스트: 10 -> 20 -> 30 -> NULL
메모리 해제 완료
```

Stack Heap

main

my_list | pointer to NODE 💀

Note: 💀 means a pointer points to memory that is either unallocated or misaligned with data boundaries. 💀 locations are *approximate* and may not match the pointer's real address. Select 'byte-level view of data' below to see more details:

C/C++ details: none [default view]

END