Program Patterns:
Linked List and (revisit) Search Pattern

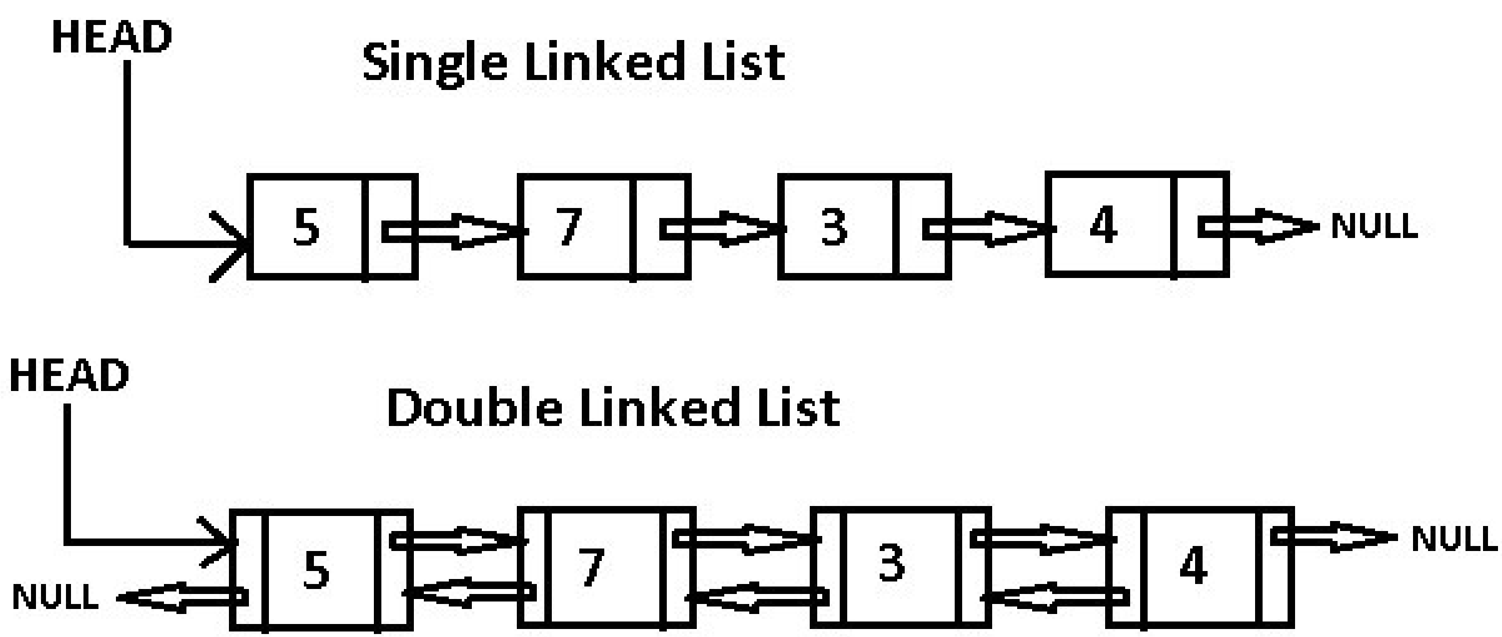
Jinwoo Choi

2024

Linked Lists

Linked List

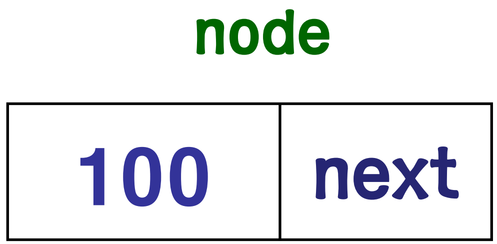
- Singly linked list
- Doubly linked list (not to be covered)



Singly Linked List

- One-way chain of data nodes
  - data node = (data,   pointer to next data)

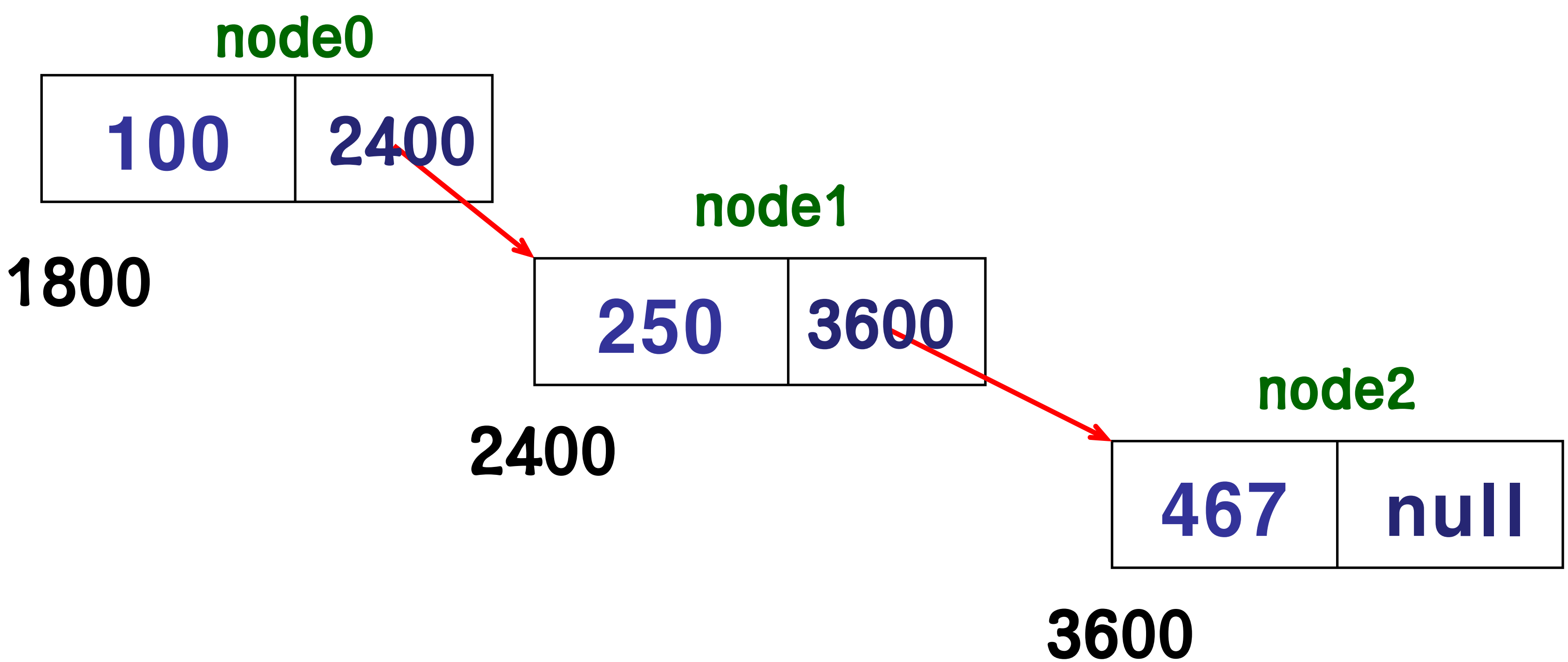
Defining a Data Node in C (Self-Referential Structure)




```
struct   NODE {
    int                  key;
    struct NODE        *next;
} node;
```

Linking the Data Nodes

```
node0.next = &node1;
node1.next = &node2;
```



A Real-World Data Node Example

node

100	Kim	25	30K	Korea	next

typedef: Define a Synonym Data Type

- typedef float mytype;

- You can use mytype instead of float.

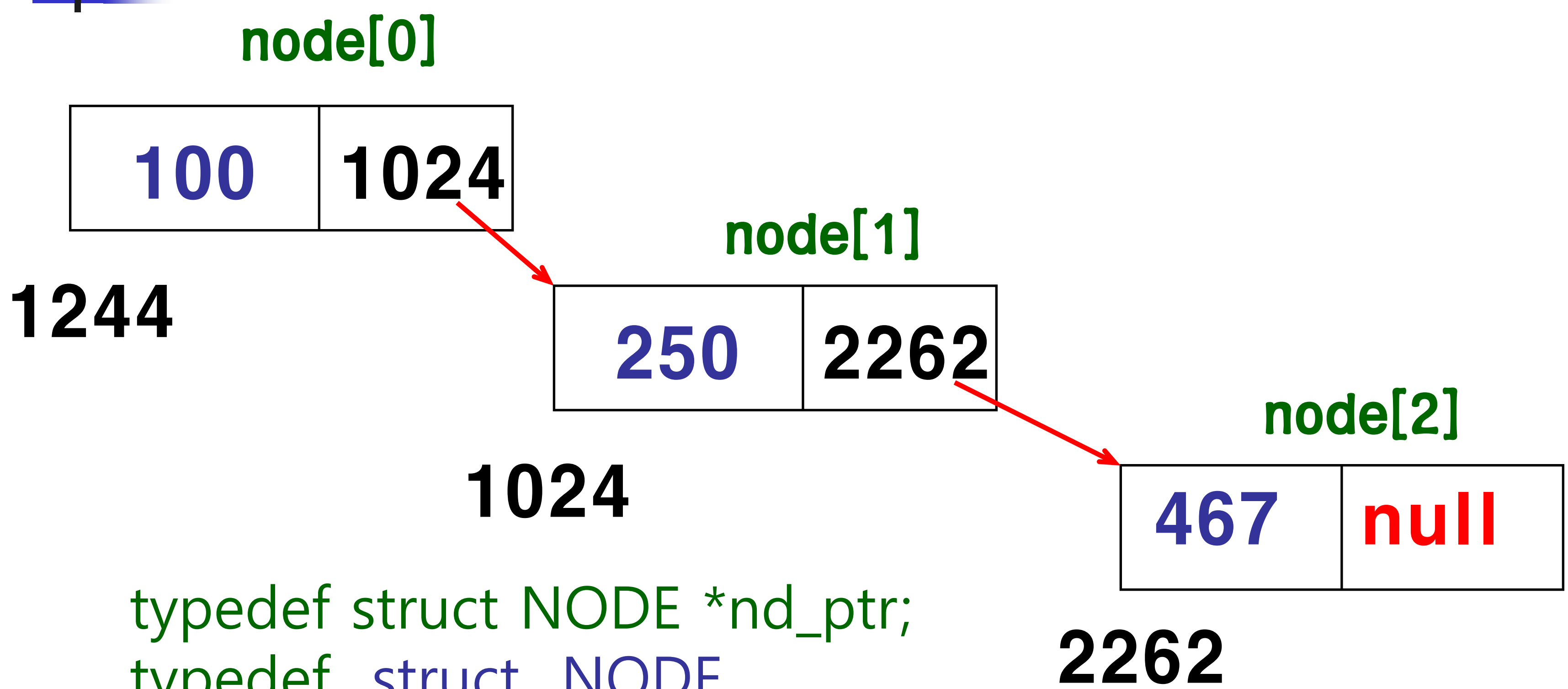
Defining a Data Node in C (Using typedef)

```
typedef  struct  NODE *nd_ptr;
typedef  struct  NODE
{
      int  key;
      nd_ptr  next;
};
```

Explained

```
typedef struct NODE *nd_ptr;
   /* data type for nd_ptr is struct NODE *   */
   /* nd_ptr can be used instead of struct NODE *   */
   /* struct NODE is defined later  */

typedef struct NODE
{
     int   key;
     nd_ptr  next;
/* nd_ptr next is the same as struct NODE *next  */
};
```

Illustrated




```
typedef struct NODE *nd_ptr;
typedef  struct  NODE
{
      int    key;
      nd_ptr   next;
} node[4];
```

Creating a Linked List

- Static node array allocation and linking
- Dynamic node allocation and linking
- Hybrid allocation and linking
    - start with a node array
    - add and link nodes dynamically

Static Memory Allocation

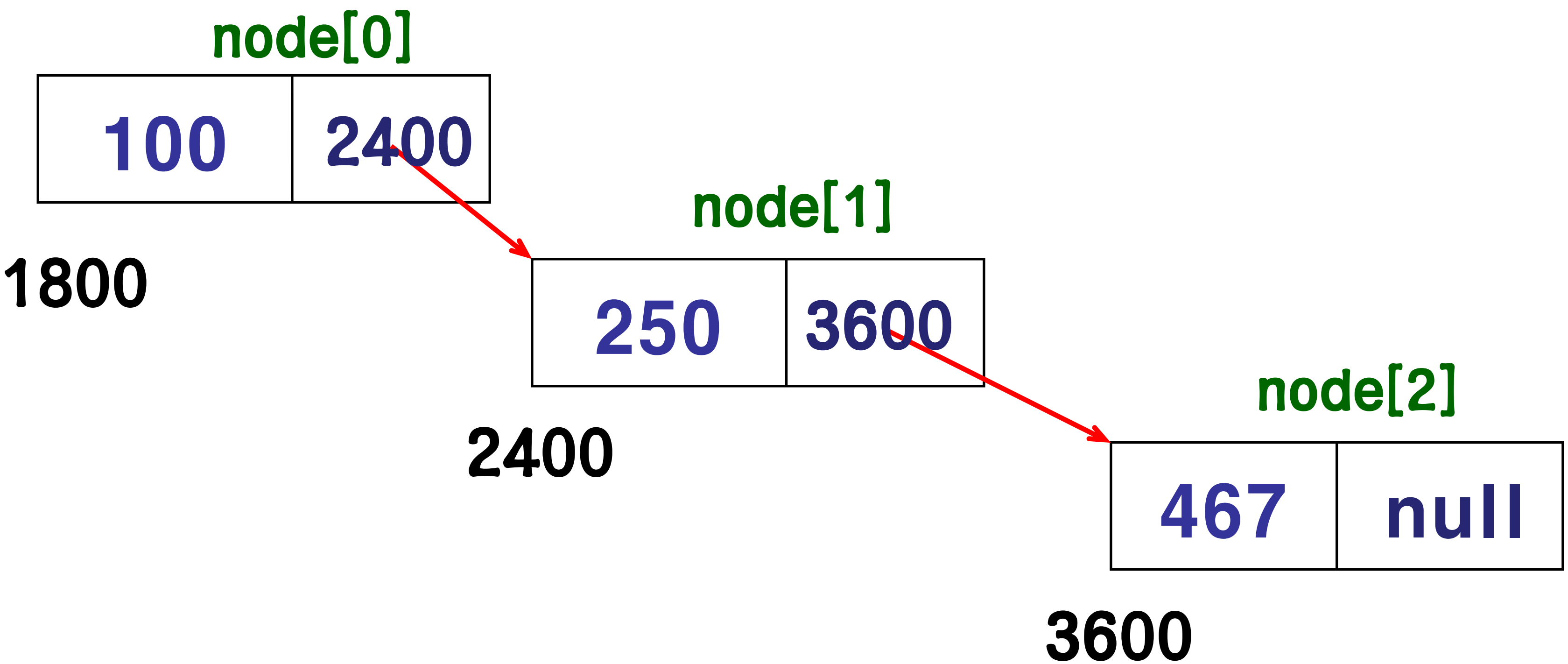
Storing Data in Data Nodes

```
node[0].key = 100;
node[1].key = 250;
node[2].key = 467;
node[0].next = node[1].next = node[2].next = NULL;
```

node[0]		node[1]		node[2]	
100	null	250	null	467	null

Linking the Data Nodes

```
node[0].next = &node[1];
node[1].next = &node[2];
```



Memory Allocation

	int	pointer	int	pointer	int	pointer	

node[0] node[1] node[2]

Dynamic Memory Allocation

Need for Dynamic Memory Allocation

- With static memory allocation, when a large amount of memory is needed, but it is difficult to determine the exact amount, to be safe, a lot of extra memory needs to be allocated.
- This can lead to a lot of memory being wasted.
- Dynamic memory allocation takes memory as needed, and in general reduces waste of memory.

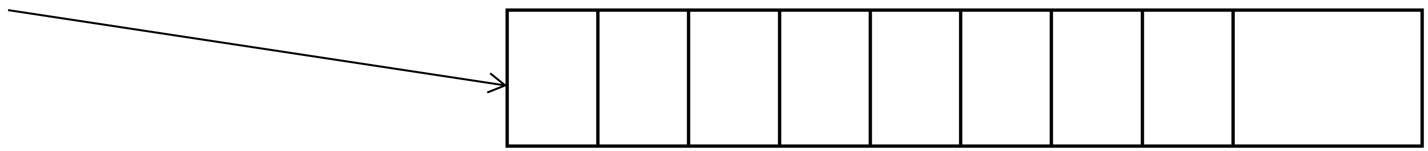
Dynamic Memory Allocation: malloc( )

```
#include <stdlib.h>
malloc (number_of_desired_bytes);
        /* function call */

void * malloc (number_of_bytes);
   /*  returns a pointer to allocated memory
     block */
   /*  or returns NULL if memory allocation
     failed  */
```

Dynamic Memory Allocation: malloc( )

23406

char *cp;



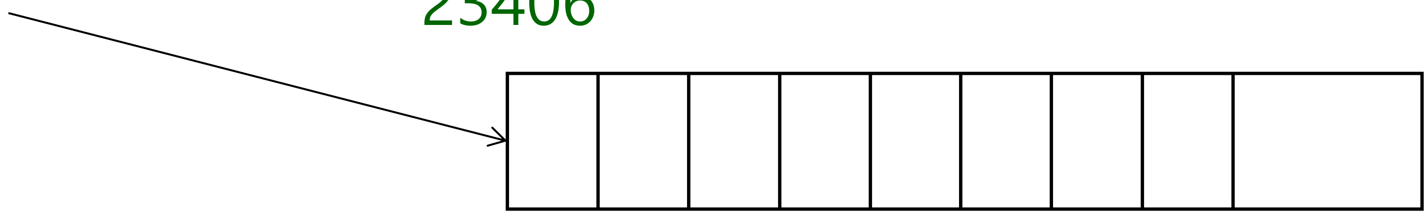
cp = (char *) malloc (1000);

/* malloc returns pointer of type void.
This needs to be type cast to desired type */

Dynamic Memory Allocation: malloc( )

int   *ip;

23406



ip = (int *) malloc (100*sizeof(int));

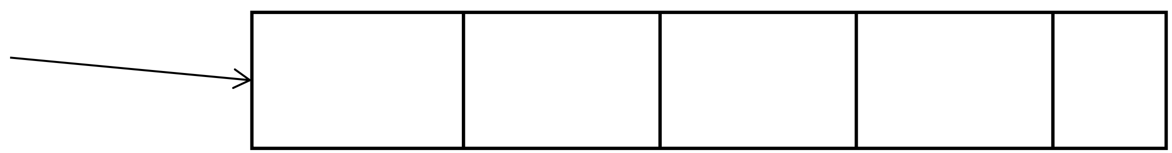
/* sizeof is a function that returns
the number of bytes for the input data type */

Dynamic Memory Allocation: malloc( )

```
struct  NODE {
    char  key[20];
    struct NODE  *next;
    };
```

23406

```
struct NODE  *link;
```



```
link = (struct NODE *) malloc (500 * sizeof(struct NODE));
```

Allocating Memory for a struct Array

```
struct  NODE {
    int   key;
    struct NODE    *next;
} node[3];
```

```
node = (struct NODE *) malloc (3*sizeof(struct NODE));
```

Dynamic Memory Deallocation: free( )

```
/* releases memory obtained by malloc  */
/* just pass the address of the memory block */
/* no need to specify the size of the memory block */

free (cp);
free (ip);
free (link);
```

Defensive Coding

- malloc may fail to allocate memory.

```
if (ip == (int *) NULL) {
    printf ("malloc failed");
    exit(1);
}
```

Exercise 1

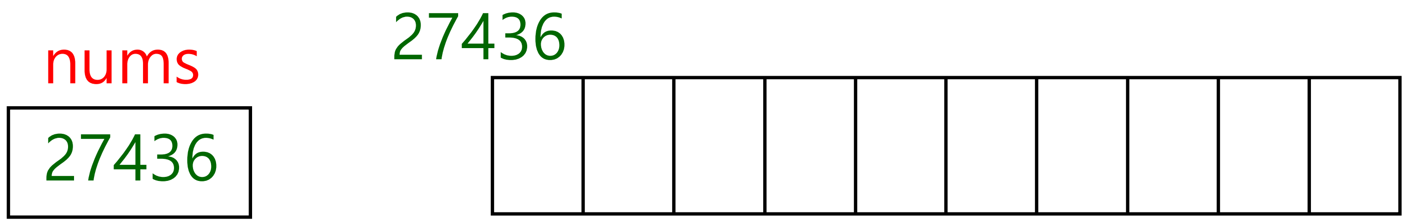
Obtain memory for an array of 10 integers, and save the starting address of the memory in an integer pointer variable int *nums

(This is equivalent to defining int nums[10].)

Free the memory obtained using malloc

Solution

```
int *nums;
nums = (int *) malloc (10*sizeof(int));
```




```
if (nums == (int *) NULL) {
    printf ("malloc failed");
    exit(1);
}

free(nums);
```

Exercise 2

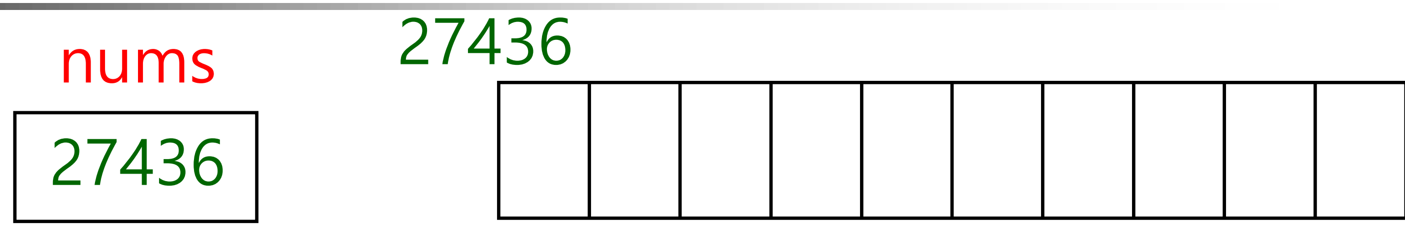
Obtain memory for an array of 10 integers, and save the starting address of the memory in an integer pointer variable int *nums

Read 10 integers and store them in the array.
(hint: store an integer in &nums[i])

Print the 10 integers in the array.

Free the memory obtained using malloc

Solution




```
int *nums, i;
nums = (int *) malloc (10*sizeof(int));
if (nums == (int *) NULL) {
    printf ("malloc failed");
    exit(1);
}

for (i=0; i<10; i++) {
    printf ("₩n type an integer")
    scanf ("%d", &nums[i]);
    printf ("%d", nums[i]);
}
free(nums);
```

Exercise 3

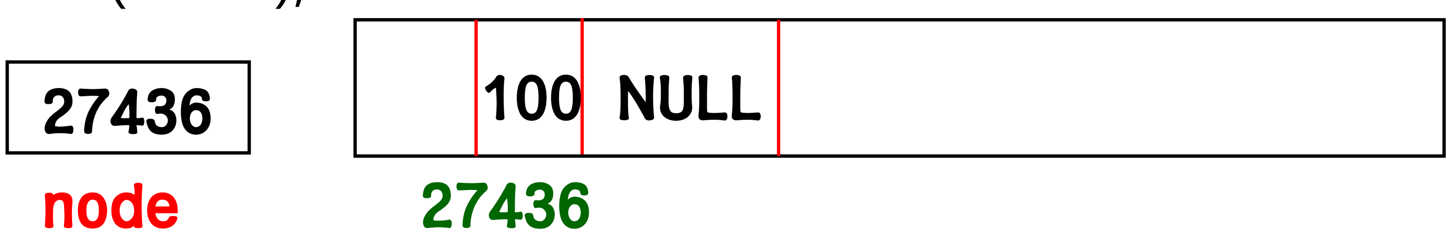
```
struct NODE {
    int key;
    struct NODE *next;
}

struct NODE *node;
```

Obtain memory for a struct NODE variable.
Save the address of the memory in variable node.
Assign an integer to key, and NULL to next in node.

Free the memory allocated.

Solution

```
struct NODE {
      int key;
      struct NODE next;
}
struct NODE *node;

node = (struct NODE *) malloc (sizeof(struct NODE));
If (node != (struct NODE *) NULL) {
      (*node).key = 100;
      (*node).next = NULL;
      free (node);
}
```



Exercise 4

```
struct NODE {
      int key;
      struct NODE *next;
}
struct NODE *node0, *node1, *node2;
```

Obtain memory for three separate struct NODE data nodes.

Assign the address of node1 to next of node0; and address of node2 to next of node1.

Assign any integer to each of the three keys.

Assign NULL to next of node2.

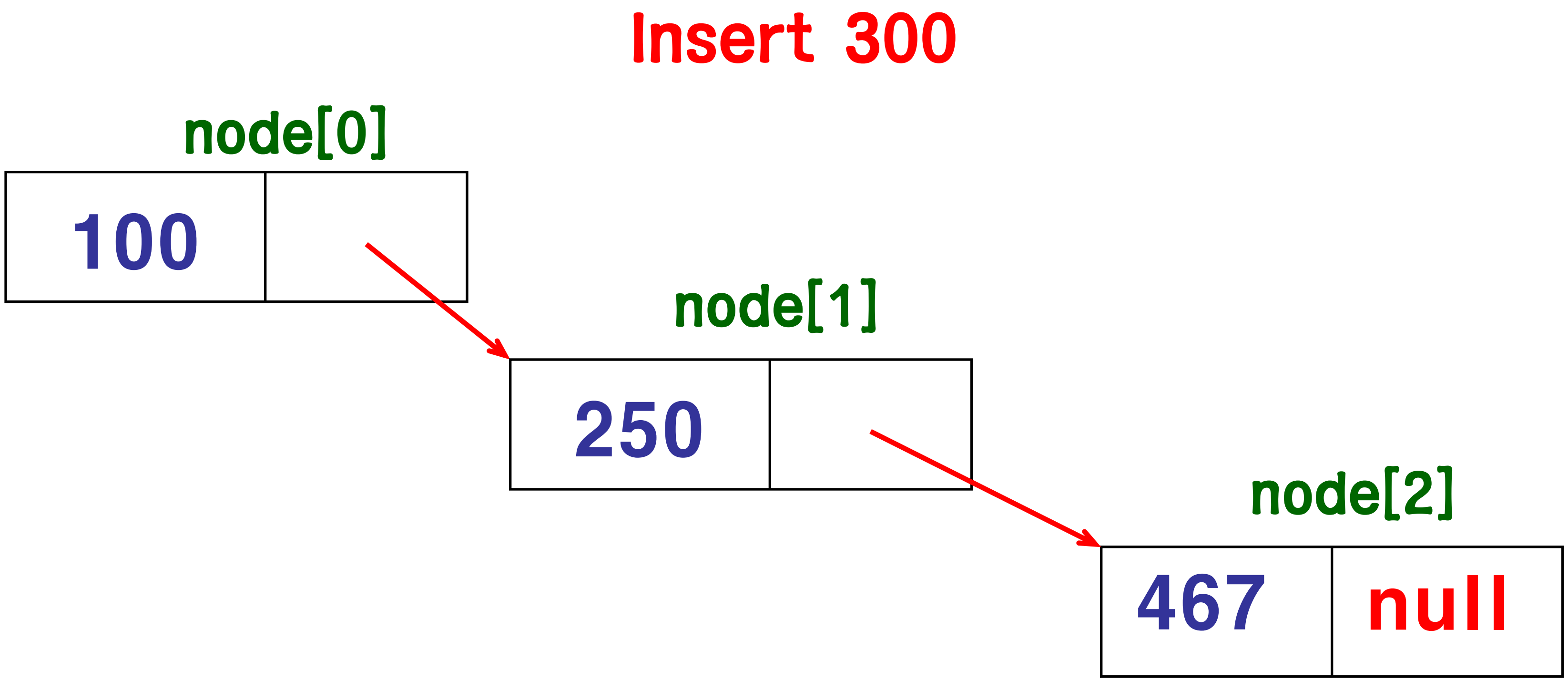
Free the memory allocated.

Operations on a Linked List

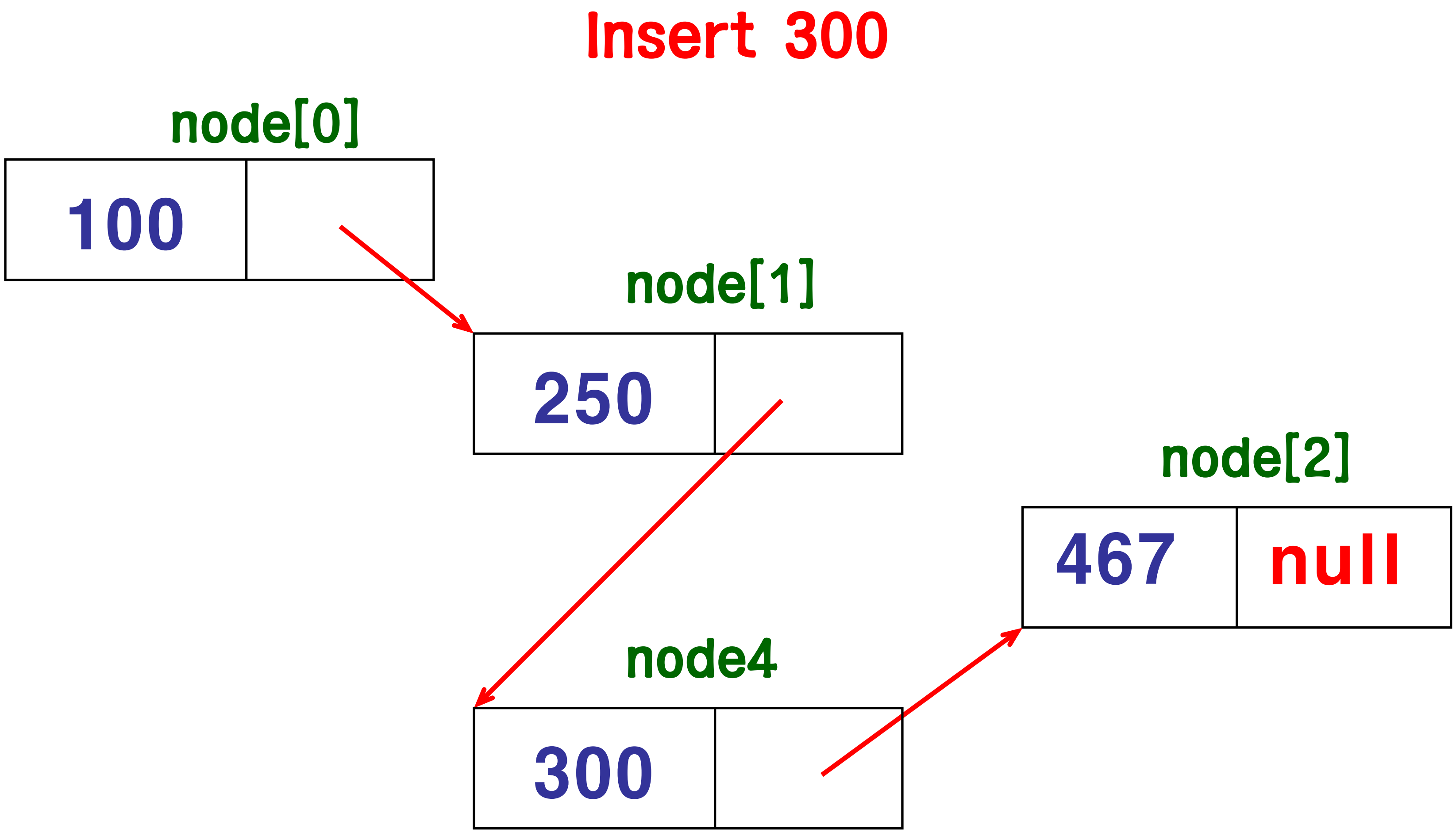
Array vs. Linked List

- Use instead of an array (when data are frequently inserted and deleted)
  - store data
  - search for data
  - insert data
  - delete data
- Use when it is necessary to maintain an order among data

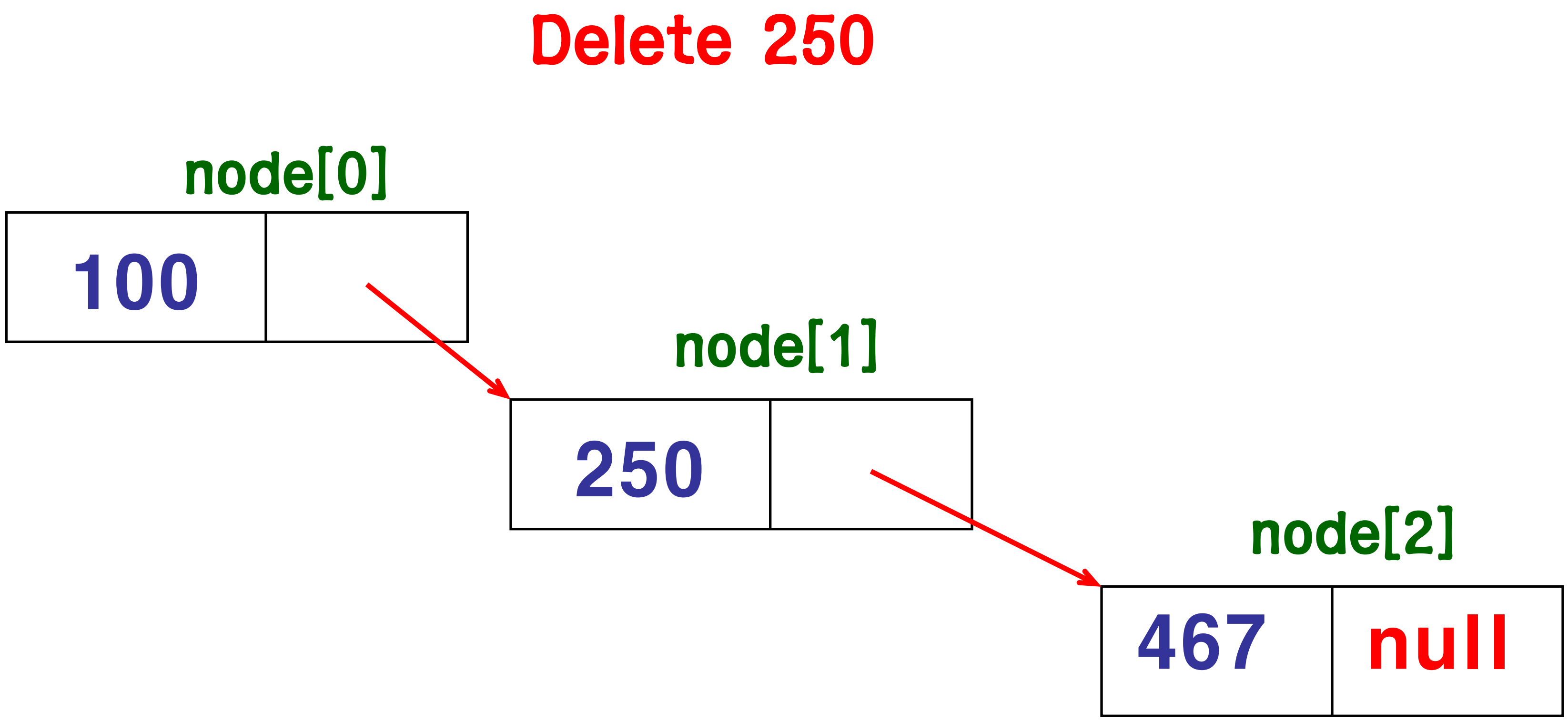
Inserting a Data Node (While Maintaining an Order)



Inserting a Data Node (While Maintaining an Order): Result (how to get this? Later)

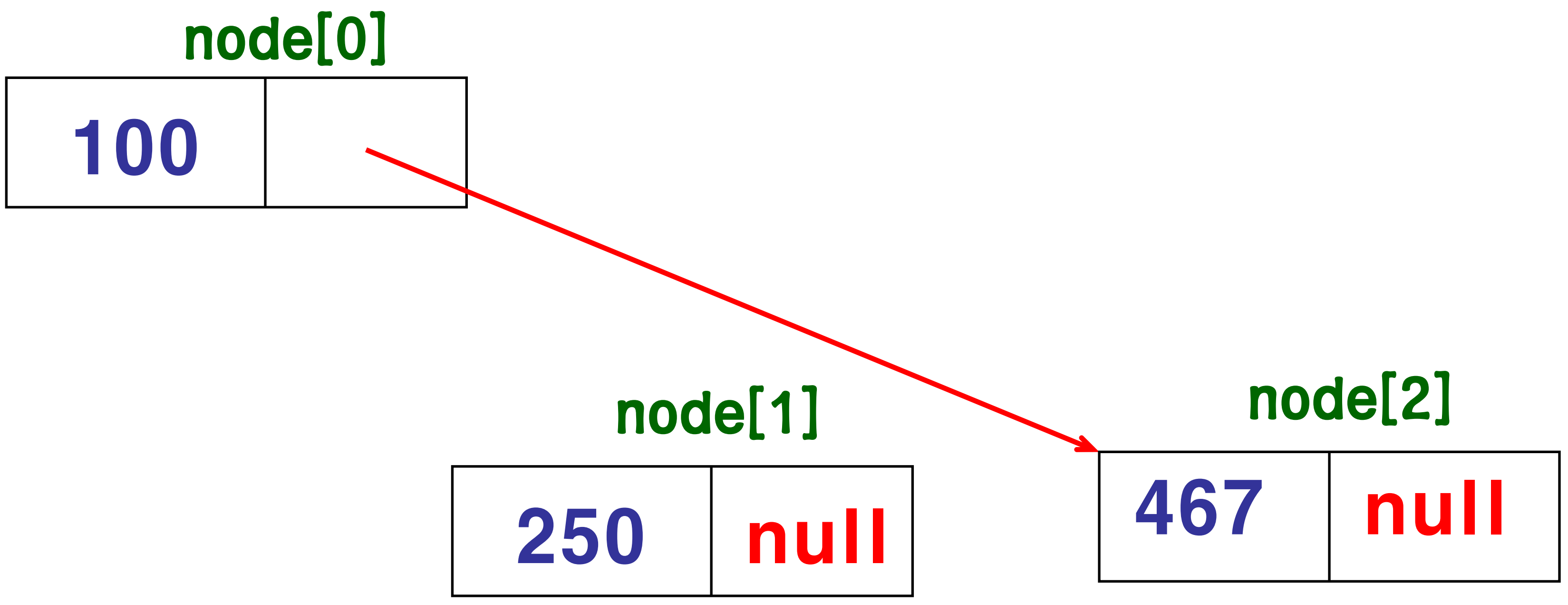


Deleting a Data Node (While Maintaining an Order)



Deleting a Data Node (While Maintaining an Order): Result (how to get this? Later)

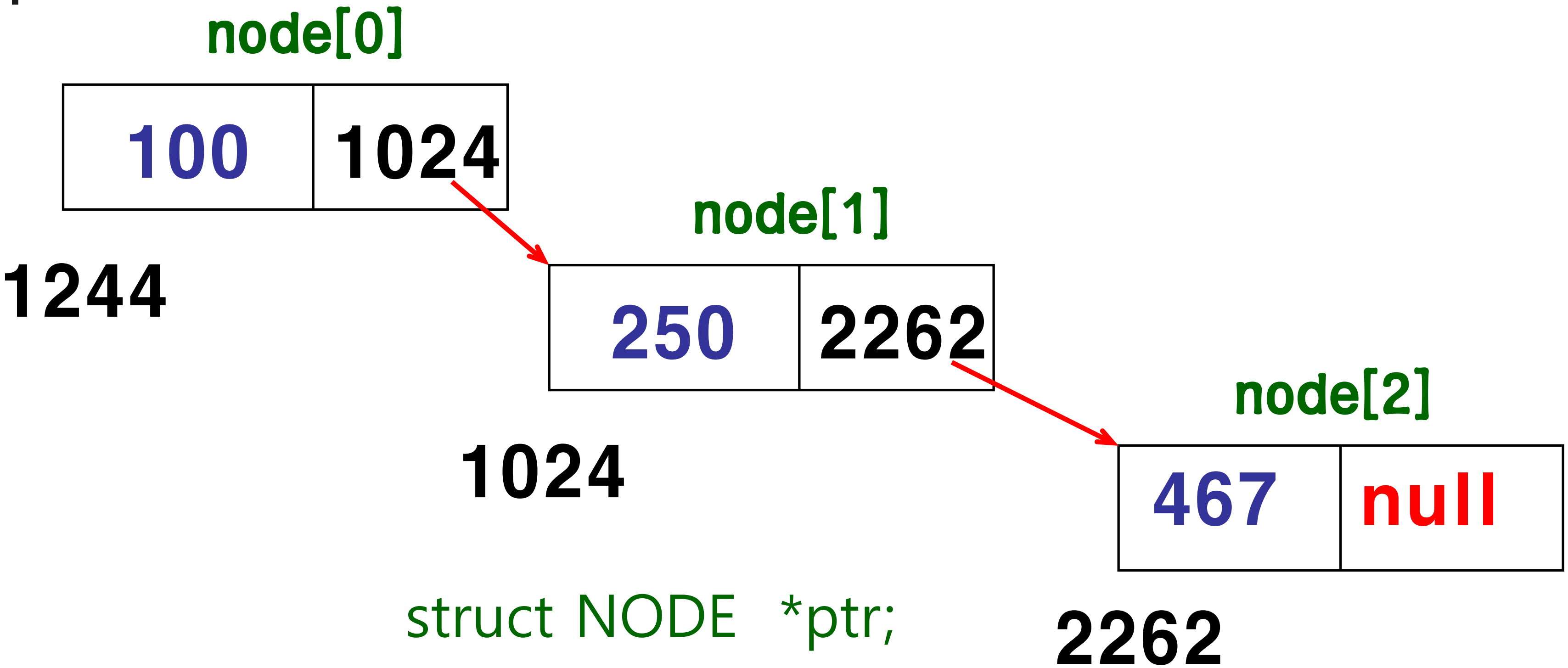
Delete 250



Searching for a Key (on a Linked List)

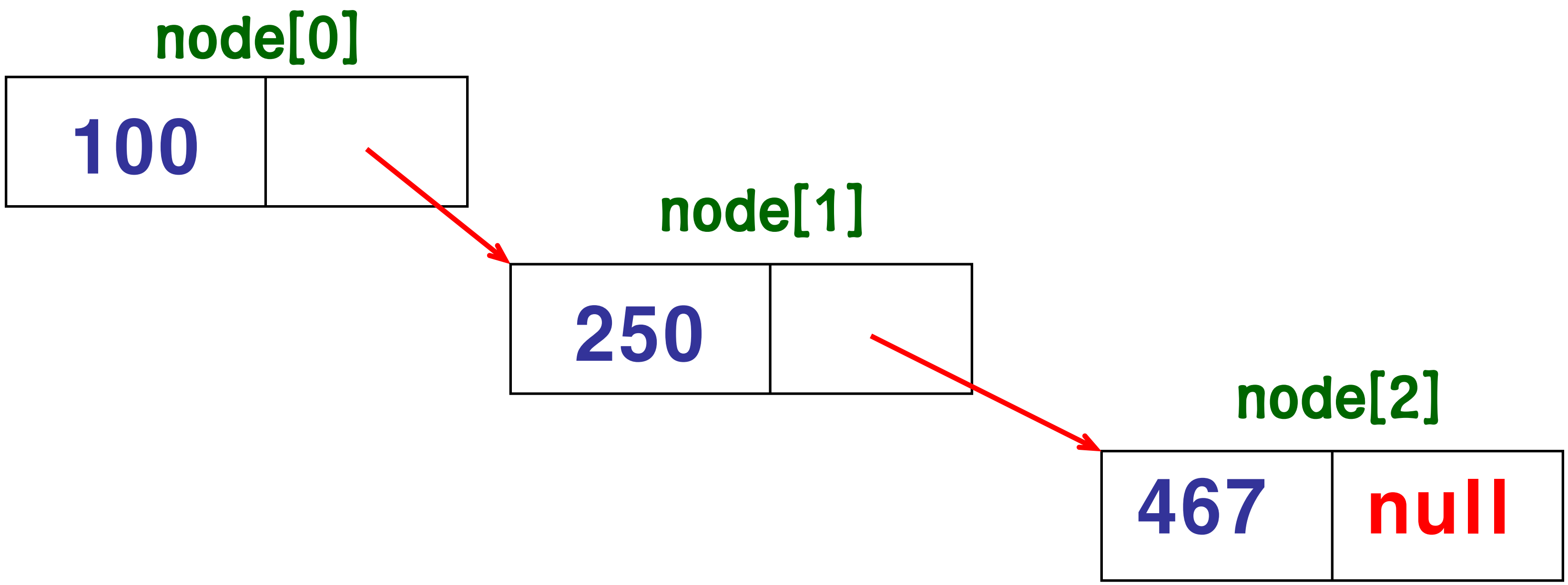
- Start with the first node (* for now *).
  - &(first_node)
- Search must stop at the last node.
  - Last node has NULL in the “next” member.

Searching for the Last Node (on a Linked List)




```
struct NODE  *ptr;

ptr = &node[0];

while (ptr != NULL)
     ptr = (*ptr).next;
```

Exercise 5: Write a C Program That Searches for a Key on a Singly Linked List

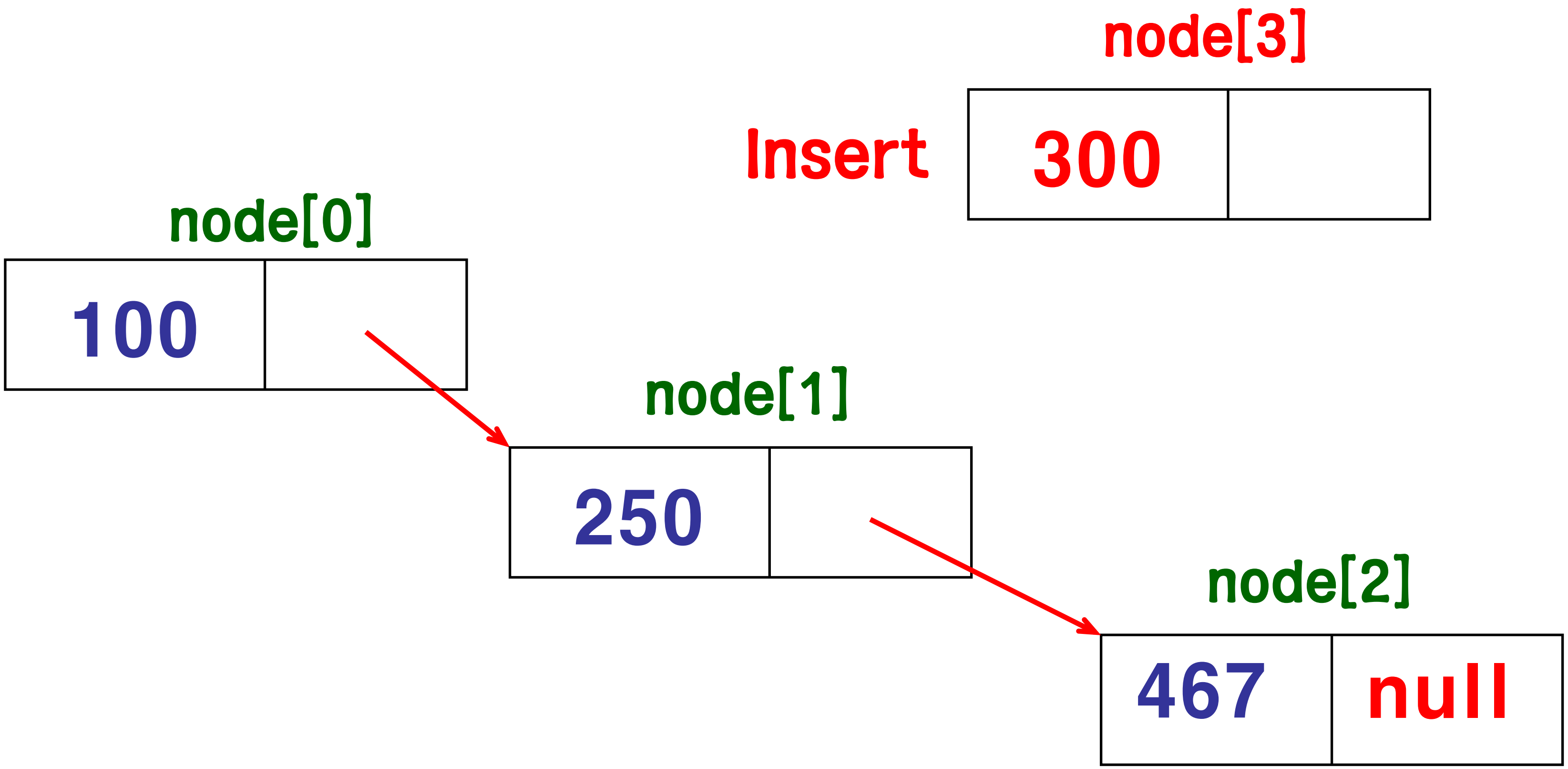


Three nodes, as shown above.
Start from node[0].
If found, print "search key found".
If not found, print "search key not found".

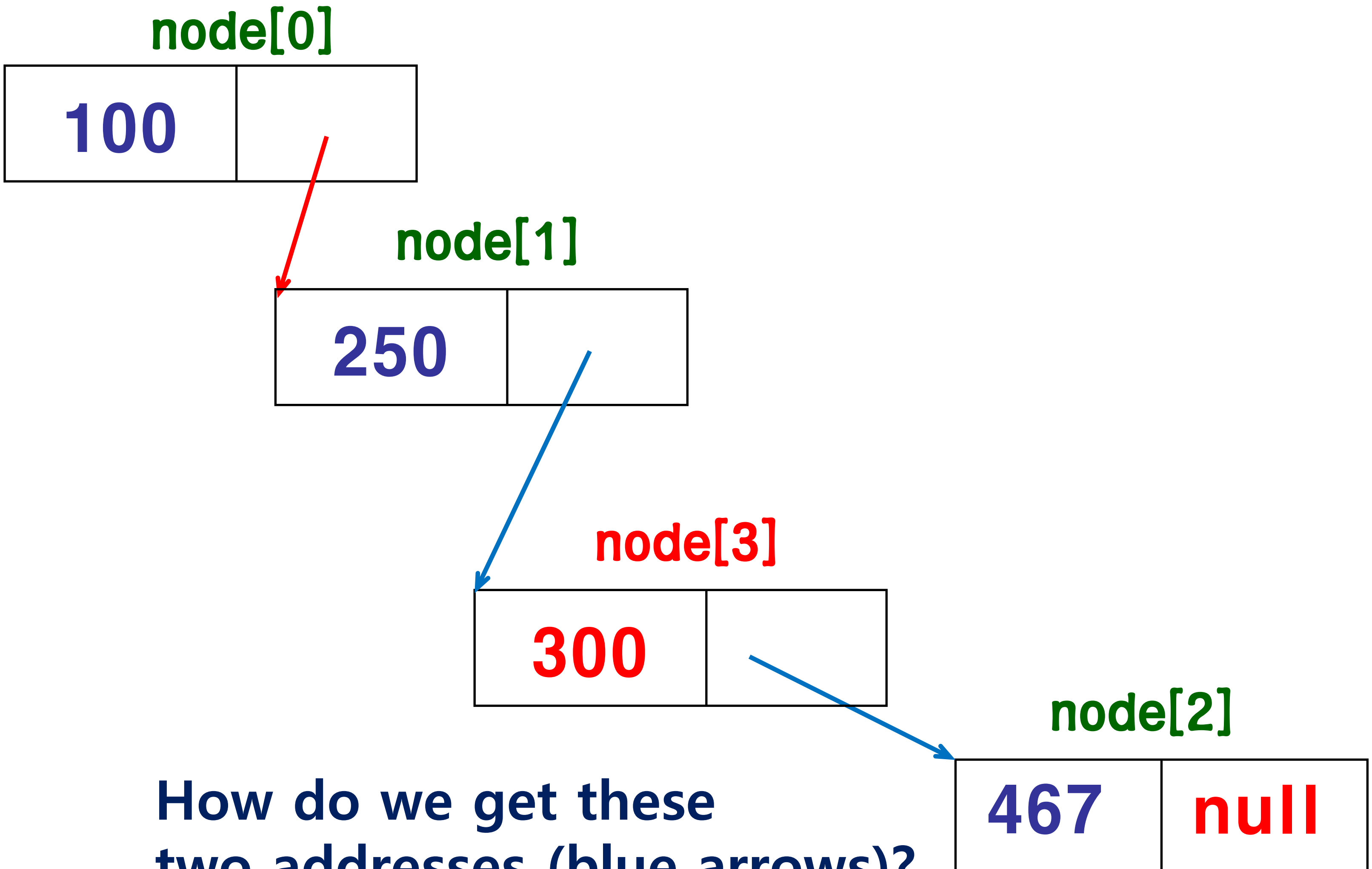
Solution

```
struct NODE  *ptr;
int   srchkey, found=0;
srchkey = 467;

ptr = &node[0];
while (ptr)
{
    if ((*ptr).key == srchkey)
    {
      found = 1;
      break;
    }
    ptr = (*ptr).next;
}
if (found)
   printf ("search key found");
else
   printf ("search key not found");
```

Exercise 6: Inserting a Node
(* for now, After the first node)



Desired Outcome



How do we get these two addresses (blue arrows)?

Step by Step (roughly)

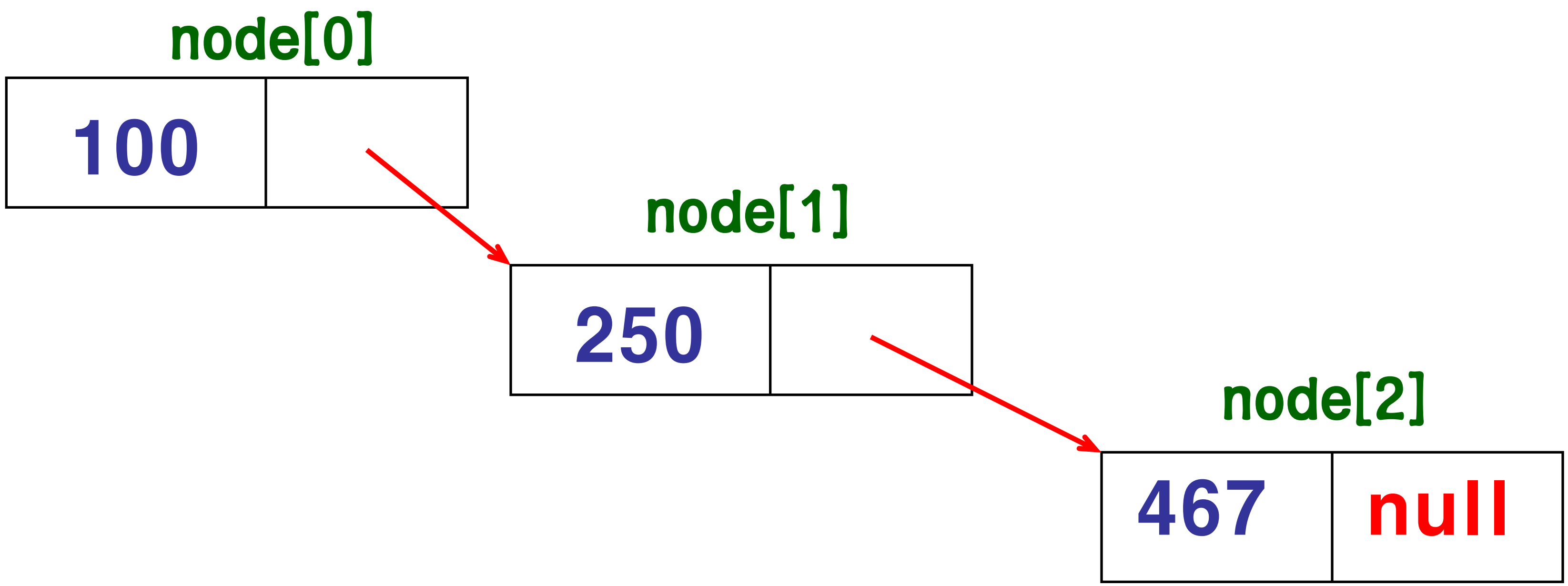
- (1) Create a new data node with key 300, and save the address of the node.
- Start the search.
- (2) Find the next node on the linked list.
- (3) Determine if the key of the current node is >300.
  - If no, save the address of the current node and go back to (2).
  - If yes, change the next pointer in the previous node to the address of the new node; and set the next pointer in the new node to the address of the current node

Solution

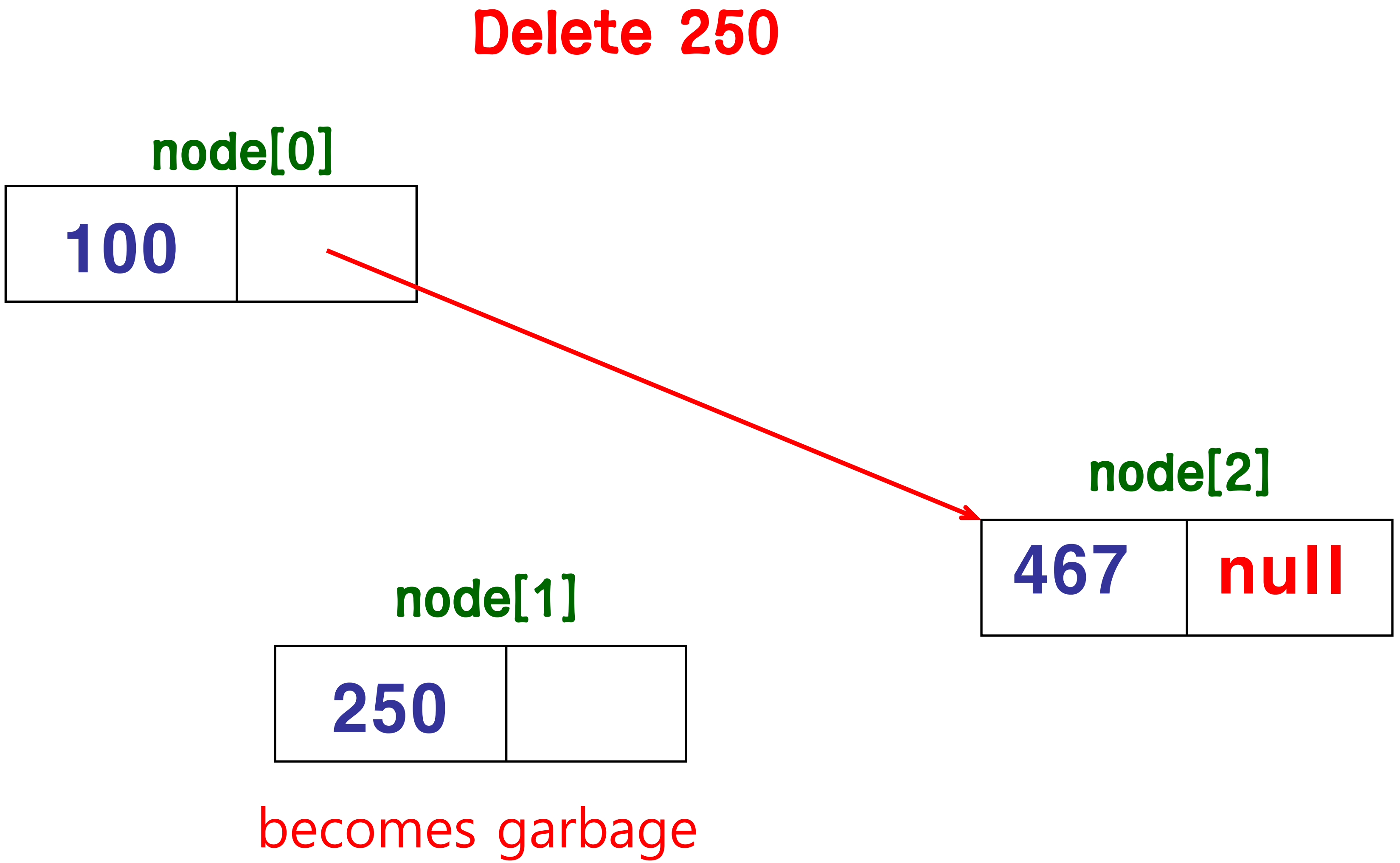
```
struct NODE  *ptr, *old_ptr=NULL;
int   newkey, fail=-1;
node[3].key = newkey = 300;
node[3].next = NULL;

/* search the linked list

ptr = &node[0];
while (ptr)
{
    if ((*ptr).key == newkey)
    {
        printf ("key already exists");
        break;
    }
    if ((*ptr).key < newkey)
    {
        old_ptr = ptr;
        ptr = (*ptr).next;
    }
    else
    {
        (*old_ptr).next = &node[3];
        node[3].next = ptr;
        printf ("key inserted");
        break;
    }

}
```

Exercise 7: Deleting a Key (Node) (for now, after the first node)

Delete 250



Desired Outcome



Step by Step (roughly)

- Start the search.
- (1) Find the next node on the linked list.
- (2) Determine if the key of the current node is 250.
  - If no, save the address of the current node and go back to (1).
  - If yes, find the next pointer in the current node. Store it as the next pointer of the previous node.

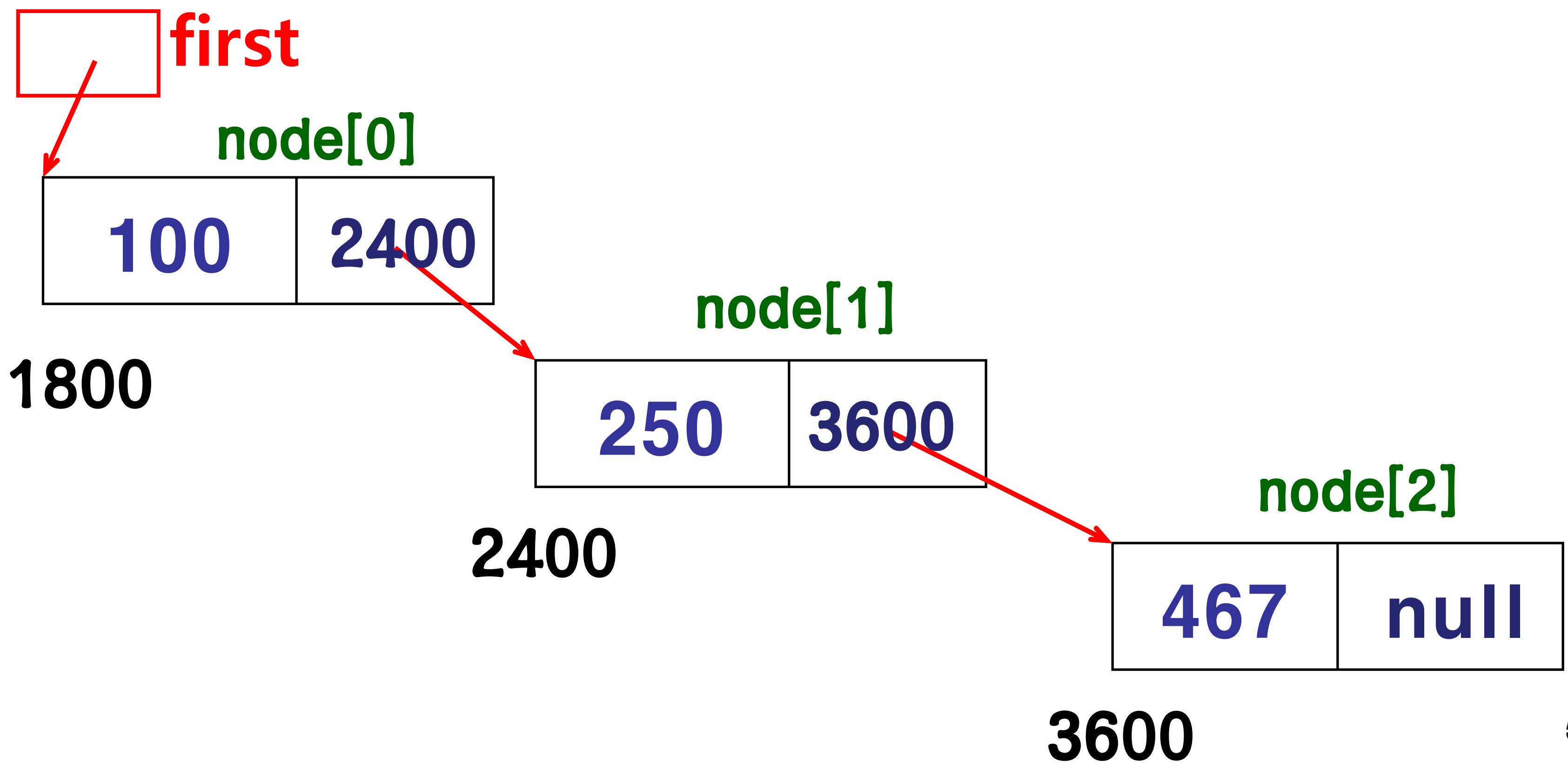
Solution

```
struct NODE  *ptr, *nx_ptr, *prv_ptr;
int   delkey, deleted=0;
delkey = 250;

ptr = &node[0];
while (ptr)
{
    if ((*ptr).key == delkey)
    {
        (*prv_ptr).next = (*ptr).next;
        (*ptr).next = NULL;
        deleted = 1;
        break;
    }
    prv_ptr = ptr;
    ptr = (*ptr).next;
}
```

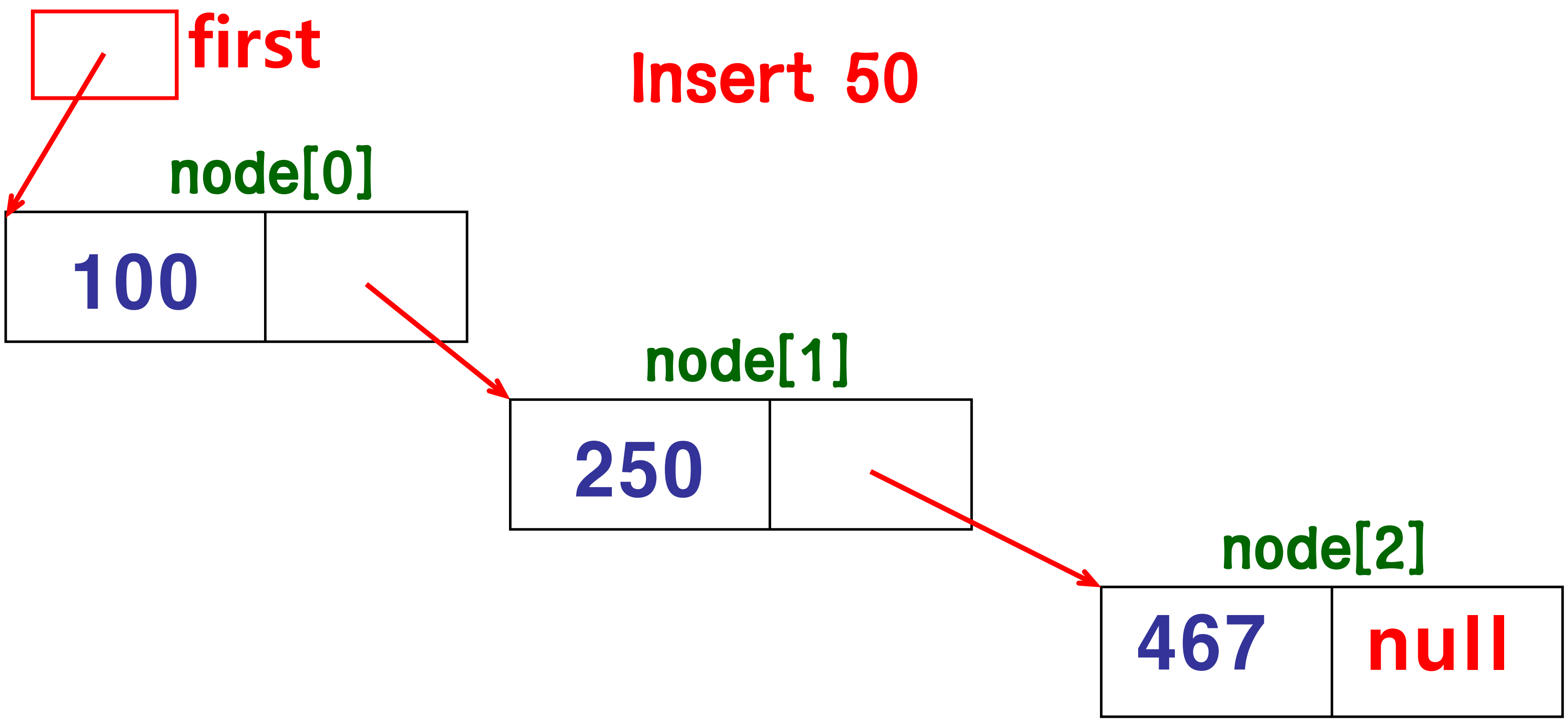
```
if (deleted)
    printf ("node deleted");
else
    printf ("key not found");
```

Searching a Linked List

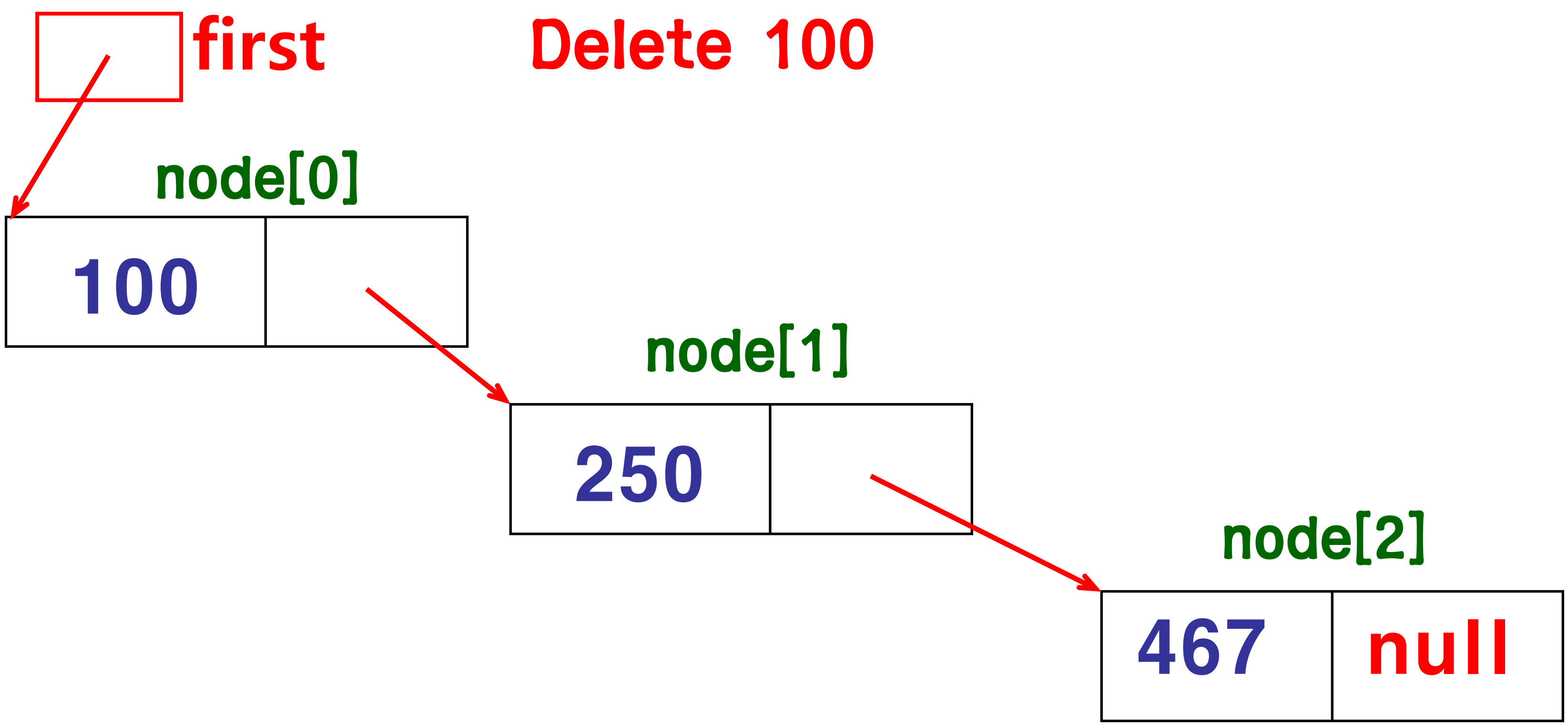
- Create a variable first (type struct NODE *) to save the address of the first node of the linked list.



Homework 1: Write a C program for inserting a new node (before the current first node), using aa function.



Homework 2: Write a C program for deleting a node (including the first node) on a linked list, using a function.

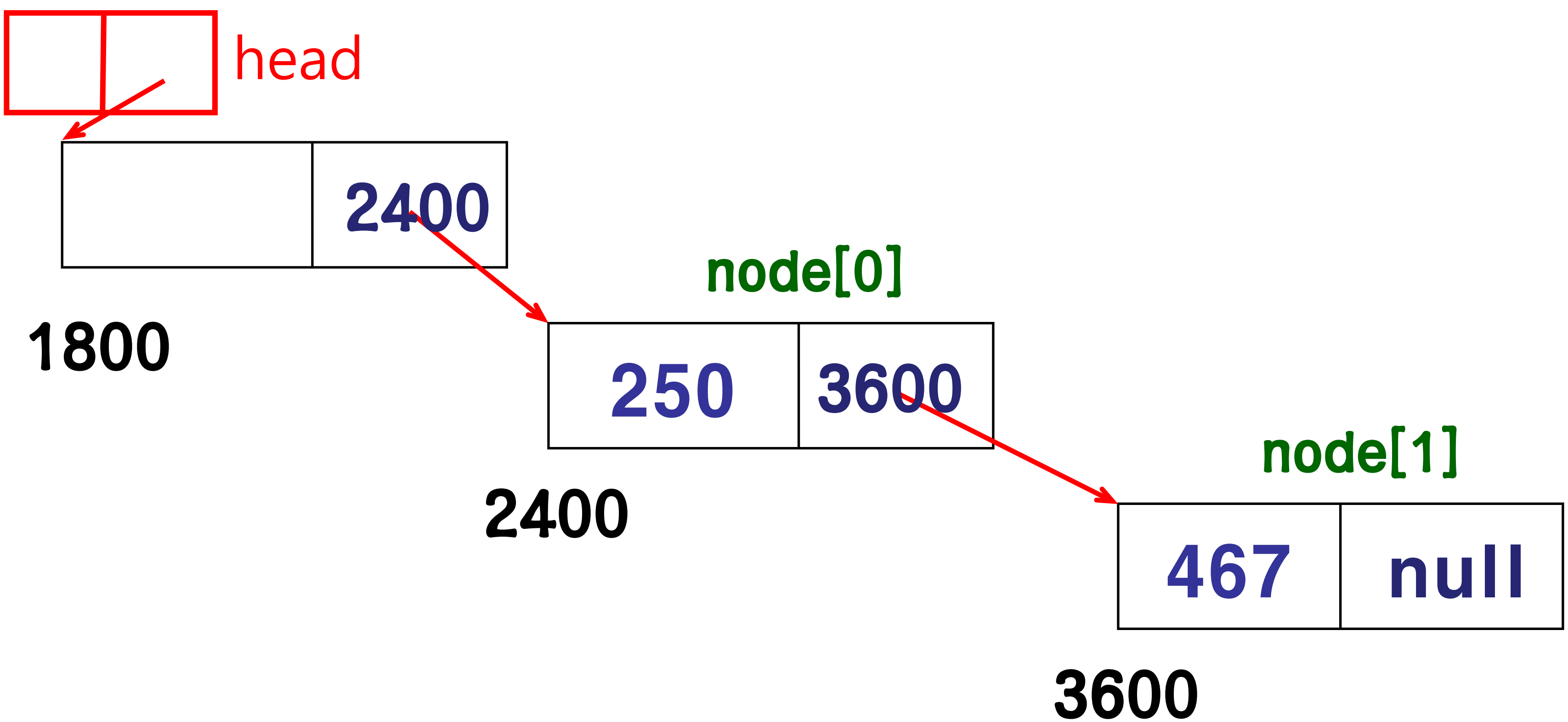


Problem with first

- Insert and delete functions become a little complex in the following cases:
  - Inserting a smaller value than the minimum value of the current linked list
  - Deleting the first node of the linked list
- Additional information about the linked list must be stored elsewhere.

Searching a Linked List: A Better Way

- Create a variable head (type struct *) to store the address of the first node of the linked list.
- head may also store information, such as the total number of nodes and the address of the last node.



Note

- The information in head must be updated, if
  - the current first node or last node is deleted, or
  - a new node is inserted as the new first node, or
  - the number of nodes on the linked list changes (due to insertion and deletion of nodes).

Creating a head

- Create a struct of struct node * type
- The next pointer in head is set to NULL

Using head to Insert a New Node: Step by Step (1/4)

Find the node to come after the new node

```
void Insert( struct NODE *head, int value )
{
   /* Start from head->next instead of head */
   struct NODE *p = head->next;

   while (p) {
       if ( p->key > value ) break;
       p = p->next;
    }
}
/* now p points to an appropriate node */
```

Using head to Insert a New Node : Step by Step (2/4)

Create a new node to insert

```
void Insert( struct NODE *head, int value )
{
   /* Start from head->next instead of head */
   struct NODE *p = head->next;
   struct NODE* new_node;

    while (p) {
          if ( p->key > value ) break;
          p = p->next;
     }
    /* create a new node  */
    new_node = (struct NODE*)malloc(sizeof(struct NODE));
    new_node->key = value;
}
```

Using head to Insert a New Node : Step by Step (3/4)

Get address of the node before the new node

```
void Insert( struct NODE *head, int value )
{
   /* Start from head->next instead of head */
   struct NODE *p = head->next, *prev = head;
   struct NODE* new_node;

    while (p) {
          if ( p->key > value ) break;
          prev = p;
          p = p->next;
     }
    new_node = (struct NODE*)malloc(sizeof(struct NODE));
    new_node->key = value;
}
```

Using head to Insert a New Node : Step by Step (4/4)

Set next pointers in before and new nodes

```
void Insert( struct NODE *head, int value )
{
   /* Start from head->next instead of head */
   struct NODE *p = head->next, *prev = head;
   struct NODE* new_node;

    while (p) {
         if ( p->key > value ) break;
         prev = p;
         p = p->next;
    }
    new_node = (struct NODE*)malloc(sizeof(struct NODE));
    new_node->key = value;
    prev->next = new_node;  /* adjust next pointers  */
    new_node->next = p;
}
```

Using head to Delete a Node : Step by Step (1/4)

Find the node to delete

```
void Delete( struct NODE *head, int value )
{
    struct NODE* p = head->next;

    while (p) {
        if ( p->key == value ) break;
        p = p->next;
    }
   /* now p points to an appropriate node */
}
```

Using head to Delete a Node : Step by Step (2/4)

Mark the node before the node to delete

```
Void Delete( struct NODE *head, int value )
{
    struct NODE *p = head->next,  *prev = head;

    while (p) {
        if ( p->key == value ) break;
        prev = p;
        p = p->next;
    }
}
```

Using head to Delete a Node : Step by Step (3/4)

Adjust the next pointer in the before node

```
Void Delete( struct NODE *head, int value )
{
    struct NODE *p = head->next,  *prev = head;

    while (p) {
        if ( p->key == value ) break;
        prev = p;
        p = p->next;
    }

    if (p)
        prev->next = p->next;  /* node deleted  */
}
```

Using head to Delete a Node : Step by Step (4/4)

Free the memory used for the deleted node

```
Void Delete( struct NODE *head, int value )
{
    struct NODE *p = head->next,  *prev = head;

    while (p) {
        if ( p->key == value ) break;
        prev = p;
        p = p->next;
    }

    if (p) {
        prev->next = p->next;  /* node deleted  */
        free( p );    /* free memory */
    }
}
```

Homework 3: Write a C program that does the following.

- In main(), create an array of 7 nodes. The keys of the first 3 nodes are 100, 250, 467 (in that order). Link them into a singly linked list (This is as done in the lecture.)
- Then, by calling the InsertKey function, insert nodes with keys 250, 300, 50, 500 (in that order).
- The linked list must maintain the keys in ascending order.
- After inserting the node with key 500, from the main program, call ScanList to traverse the linked list from the first node to the last node, and print the key of each node in sequence.
- InsertKey has three parameters:
  - new key to be inserted.
  - head of the linked list.
  - address of the new first node on the linked list, if a new first node was created. (If no new first node was created, the address returned is NULL.)
- The return type of InsertKey is int.
  - 0 if insert was successful.
  - -1 if insert was not successful (key already exists).

Homework 4: Write a C program that does the following:

- In main(), create an integer array int nums[10] and initialize it with (17, 39, 11, 9, 42, 12, 15, 8, 13, 41).
- Then, call a function to convert this array into a linked list of struct nodes

  struct NUM { int key; struct NUM *next};

  by copying the integer from the array in sequence

  (i.e., from nums[0], nums[1],…nums [9])
- On the linked list, the keys are in ascending order.
- In main(), print all keys on the linked list.

(Revisiting) Search Patterns

Data Search

- search for max/min
- element count
- Boolean predicate-based search
- string match
  - exact match
  - partial match
  - approximate match

Data Search Patterns

- Predicate-based search
- Exact match
- Partial match

Predicate-Based Search

- Limit the search space using Boolean predicates (search conditions).
  - multiple predicates using AND/OR

Predicate-Based Search

- Stored data

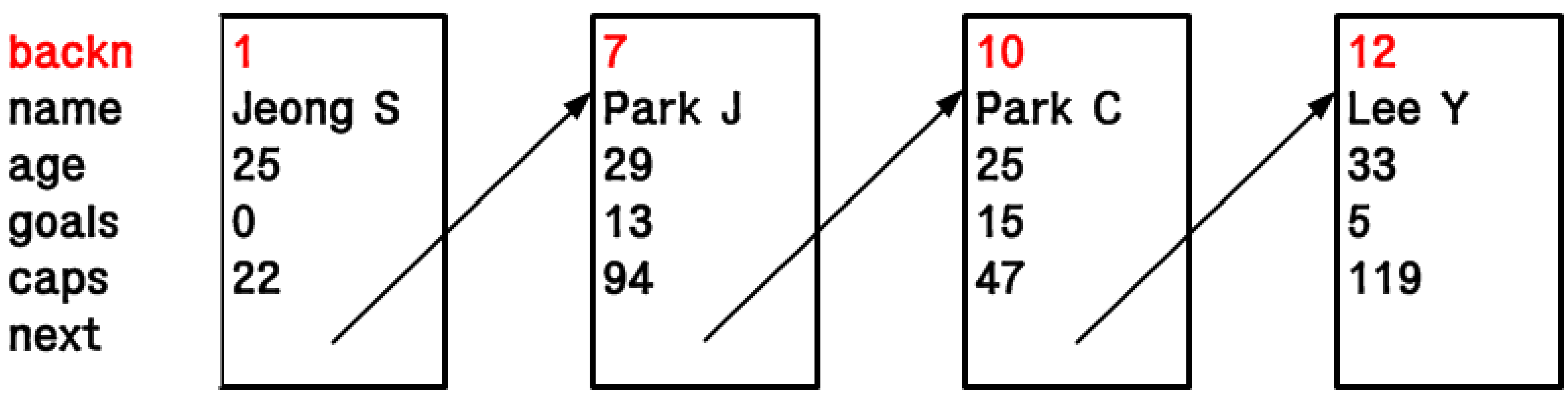
Back Number (sorted)	Name	Age	A-matches	Goals
1	Jung Sung-Ryong	25	22	0
7	Park Ji-Sung	29	94	13
10	Park Chu-Young	25	47	15
12	Lee Young-Pyo	33	119	5
16	Ki Sung-Yueng	21	28	4
17	Lee Chung-Yong	22	27	4
22	Cha Du-Ri	30	51	4

Exercise 8: Problem

- Read data about 5 players (back number, name, age, goals, and A-matches), and store them in nodes of a linked list, sorted by the "back number" in ascending order.
- Next, read search conditions.
  - minimum age
  - maximum age
  - minimum goals
  - maximum goals
  - minimum A-matches
  - maximum A-matches
- Then, print the player data for the players who satisfy the conditions. Print the player data in the following format
  - back number, name, age, goals, A-matches

Step 1: Understand the problem

- Create an example linked list
  - search for the players who satisfy the search conditions
- Create an example query (search conditions)
  - find a player
    - age is 25-29, and
    - scored at least 5 goals, and
    - played at least 30 A-match games



Step 2: Outline a Solution

- Read data about 5 players, and save them on a singly linked list.
- Read search conditions.

- Search each node of the linked list and see if the player data satisfy all of the search conditions.
  - The search conditions should be ANDed.
- Print the player data for each player who satisfies the search conditions.

Step 3: Form a Program Structure

- Read data about 5 players, and save them on a singly linked list.
- Read search conditions.

- Search the linked list for players who satisfy all the search conditions.
- Print the player data for each player who satisfies all the search conditions.

Step 4: Write Pseudo Code

```
for loop (1 through 5)
   read data from user
   store in a linked list sorted by the back number

read search conditions

/* search the linked list for match */
while (node next pointer is not NULL)
    see if all search conditions are TRUE
    if TRUE
       print the player data
    move to the next node
```

End of Class