Chapter 6
Structures
스트럭처

Part 2
6.6~6.9

CSE2018 시스템프로그래밍기초
2016년 2학기

한양대학교 ERICA
컴퓨터공학과 => 소프트웨어학부
도경구

1. Basics of Structures
2. Structures and Functions
3. Arrays of Structures
4. Pointers to Structures
5. Self-referential Structures
6. Table Lookup
7. Typedef
8. Unions
9. Bit-fields

Table Lookup

Problem

테이블에 기록

```
#define IN 1
```

기록된 문자열로 대치

```
state = IN;
```

↓

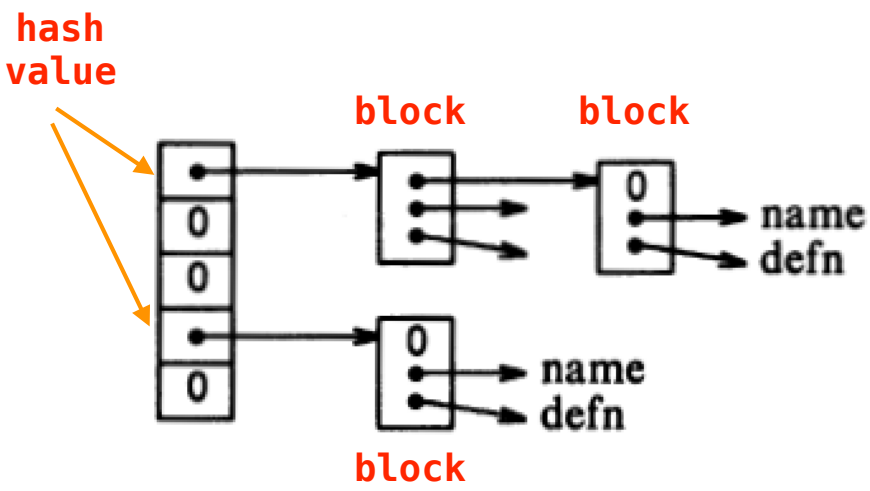
```
state = 1;
```

Design

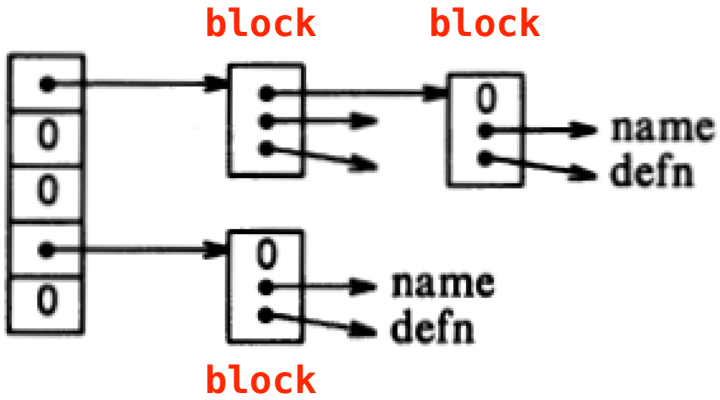
`install(s, t)`	이름 s과 대치문자열 t를 테이블에 기록함
`lookup(s)`	데이블에서 이름 s를 찾아 포인터를 내줌, 없으면 NULL을 내줌

Algorithm : Hash

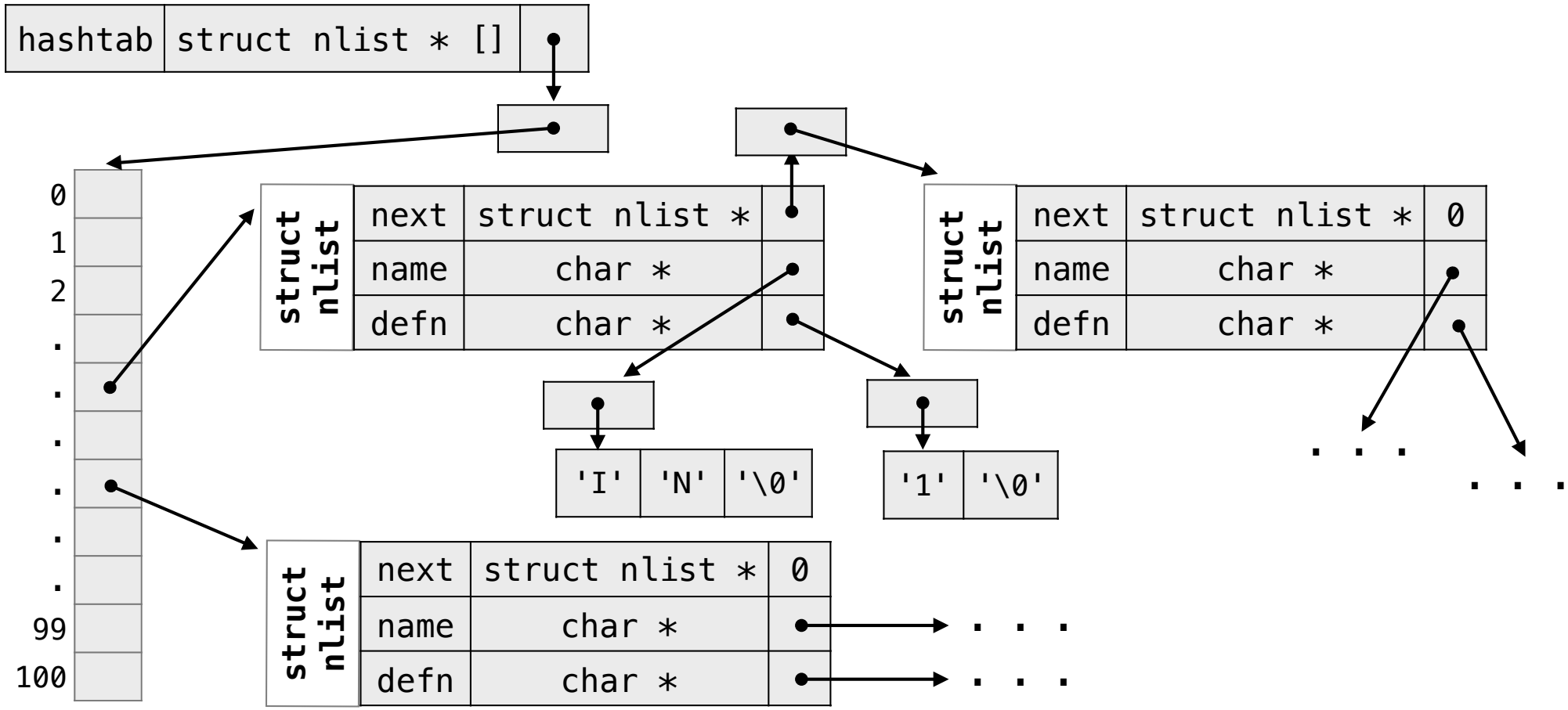
- 이름을 해시값(0포함한 양의 정수)으로 변환하여 포인터 배열의 인덱스로 사용
- 배열의 포인터는 해당 해시값 소속 이름 정보를 갖고 있는 블록 리스트의 앞부분을 가리킴
- 해시값에 소속한 이름 정보가 없으면 NULL(0)로 표시



```
struct nlist {
    struct nlist *next;
    char *name;
    char *defn;
};
```



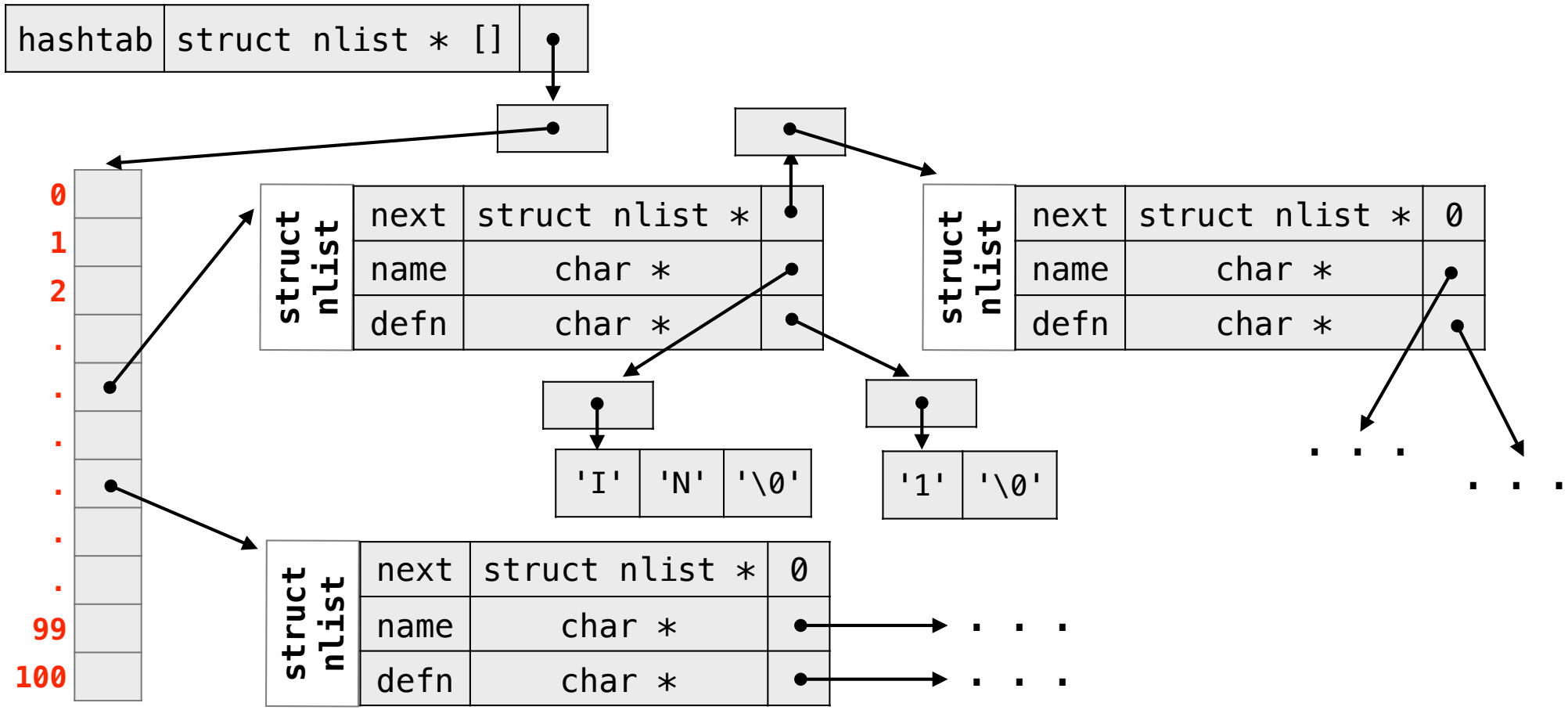

```
#define HASHSIZE 101

static struct nlist *hashtab[HASHSIZE];
```



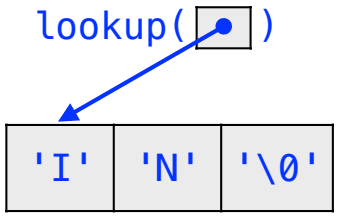
해시값 계산하기

해시값 계산속도는
빠를 수록 좋다.

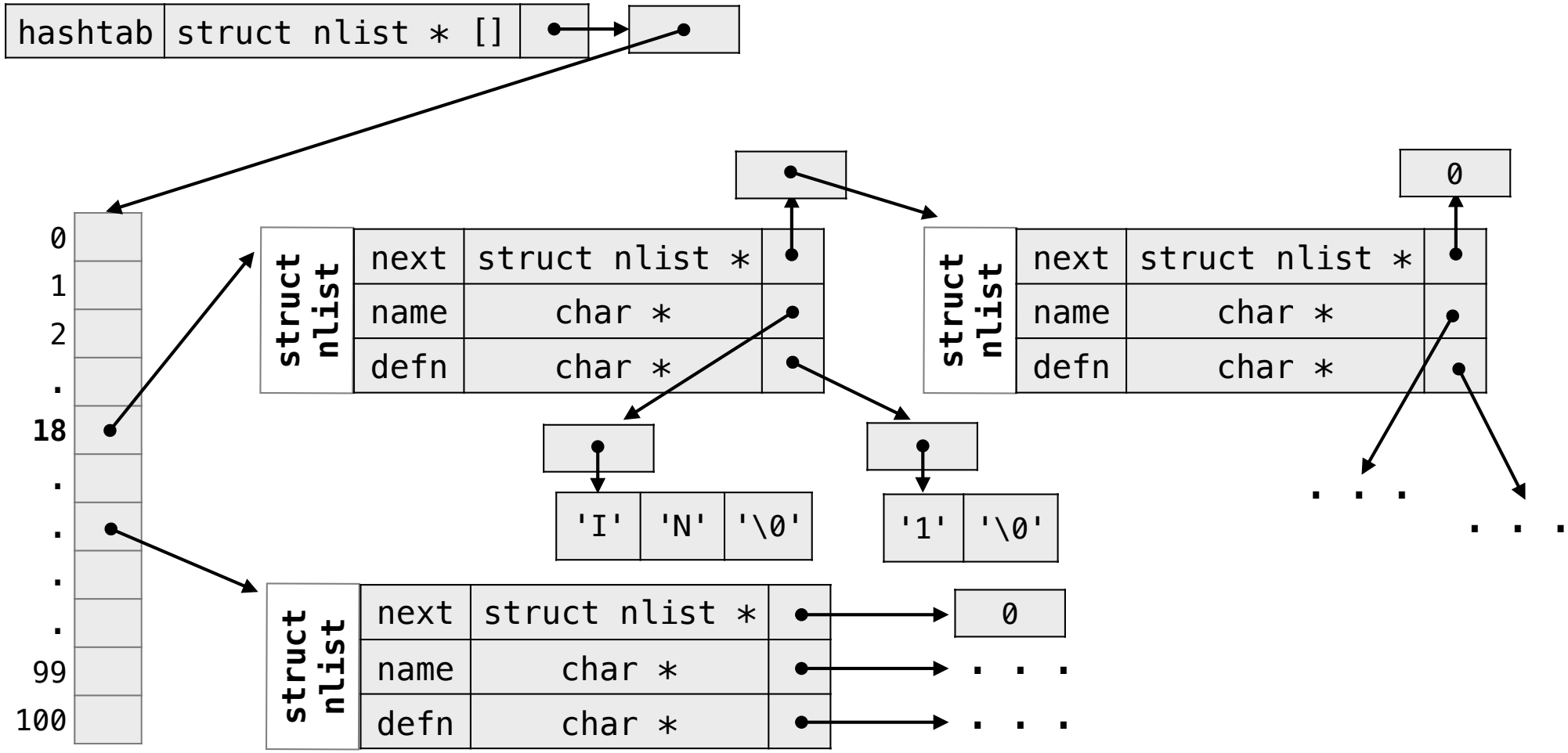
```
/* hash: form hash value for string s */
unsigned hash(char *s) {
    unsigned hashval;

    for (hashval = 0; *s != '\0'; s++)
        hashval = *s + 31 * hashval;
    return hashval % HASHSIZE;
}
```



찾기

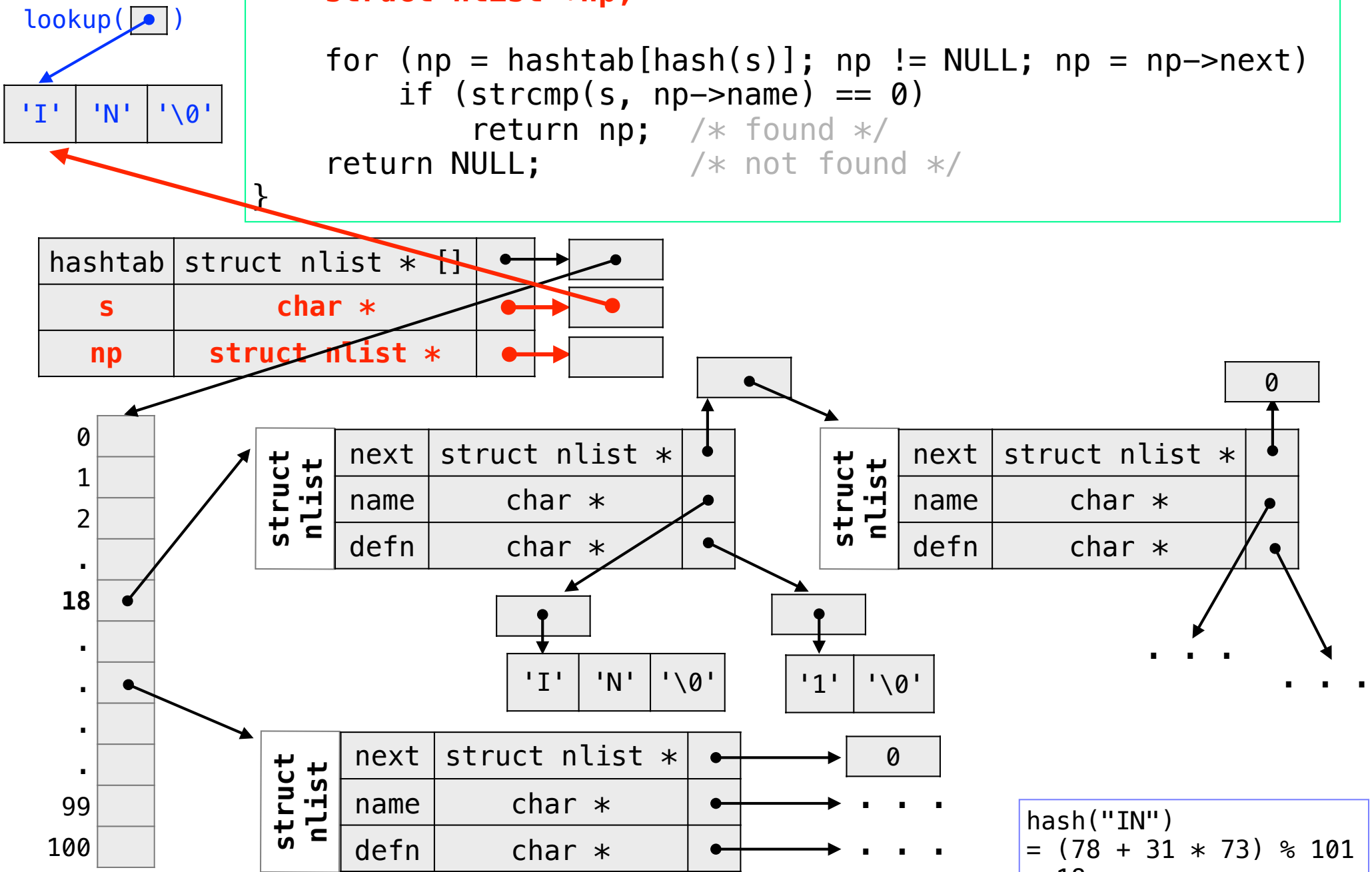



```
/* lookup: look for s in hashtab */
struct nlist *lookup(char *s) {
    struct nlist *np;

    for (np = hashtab[hash(s)]; np != NULL; np = np->next)
        if (strcmp(s, np->name) == 0)
            return np;  /* found */
    return NULL;        /* not found */
}
```



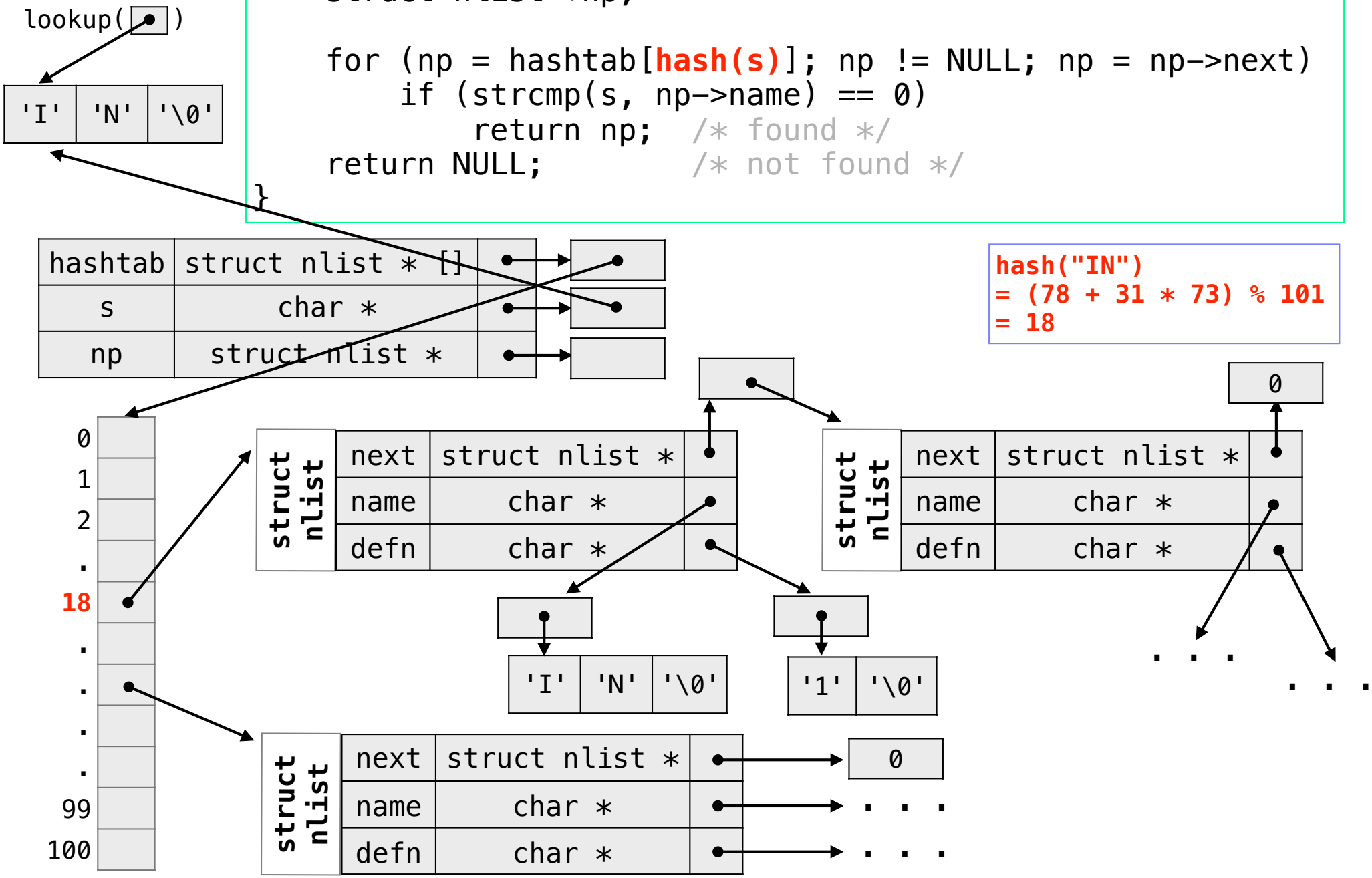
찾기

```
/* lookup: look for s in hashtab */
struct nlist *lookup(char *s) {
    struct nlist *np;

    for (np = hashtab[hash(s)]; np != NULL; np = np->next)
        if (strcmp(s, np->name) == 0)
            return np;   /* found */
    return NULL;         /* not found */
}
```



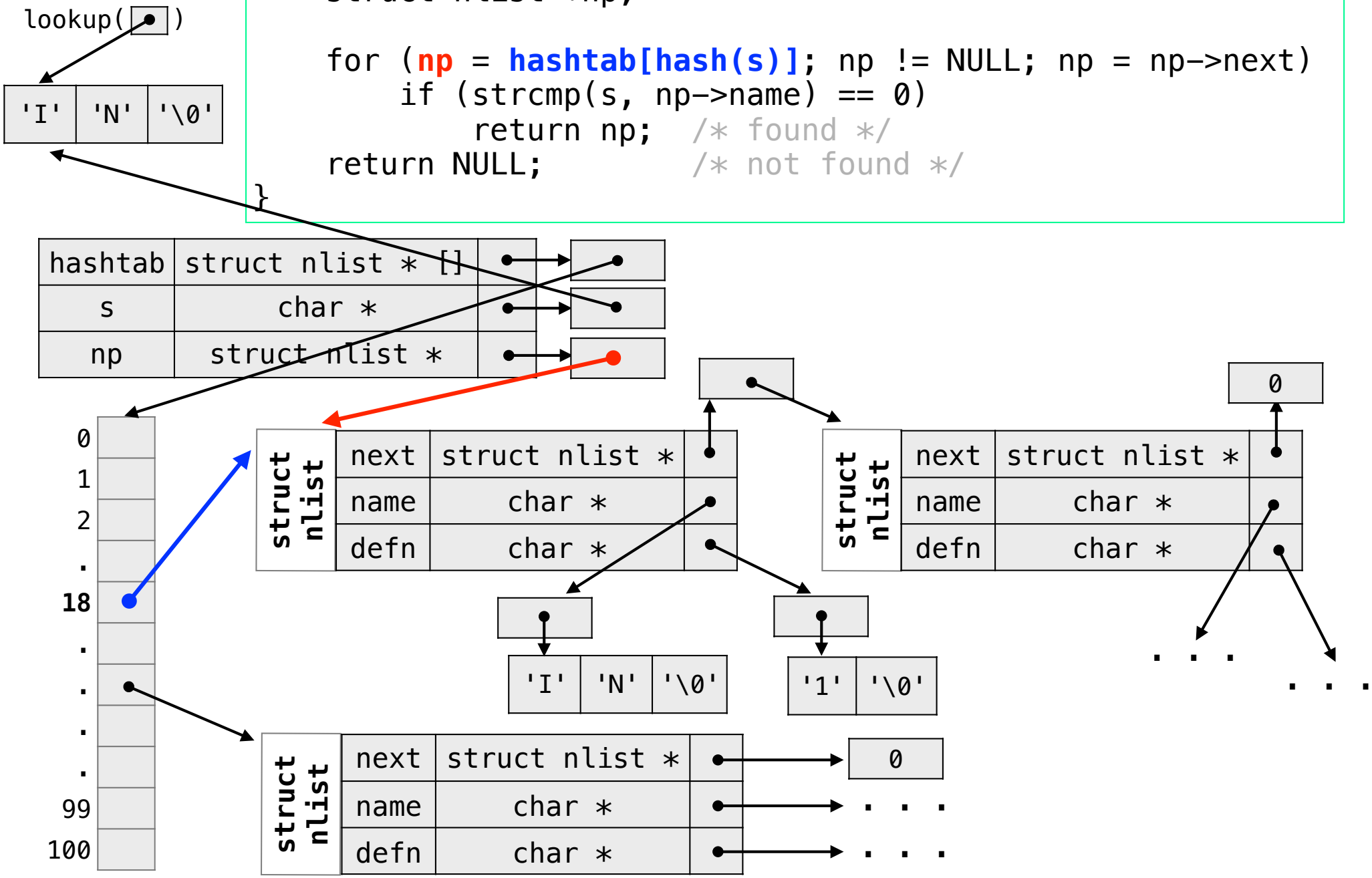
찾기

```
/* lookup: look for s in hashtab */
struct nlist *lookup(char *s) {
    struct nlist *np;

    for (np = hashtab[hash(s)]; np != NULL; np = np->next)
        if (strcmp(s, np->name) == 0)
            return np;  /* found */
    return NULL;        /* not found */
}
```



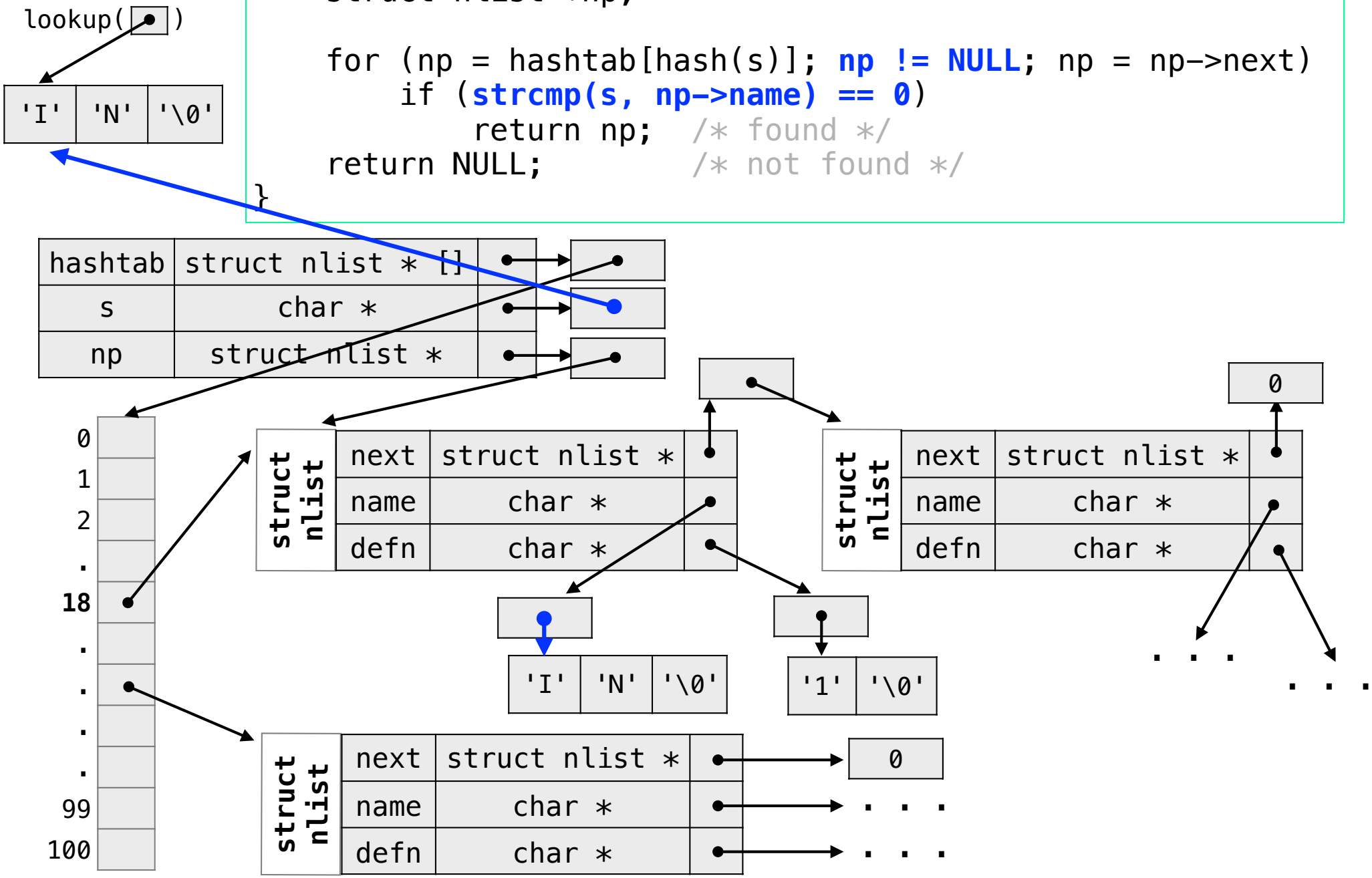
찾기

```
/* lookup: look for s in hashtab */
struct nlist *lookup(char *s) {
    struct nlist *np;

    for (np = hashtab[hash(s)]; np != NULL; np = np->next)
        if (strcmp(s, np->name) == 0)
            return np;  /* found */
    return NULL;        /* not found */
}
```



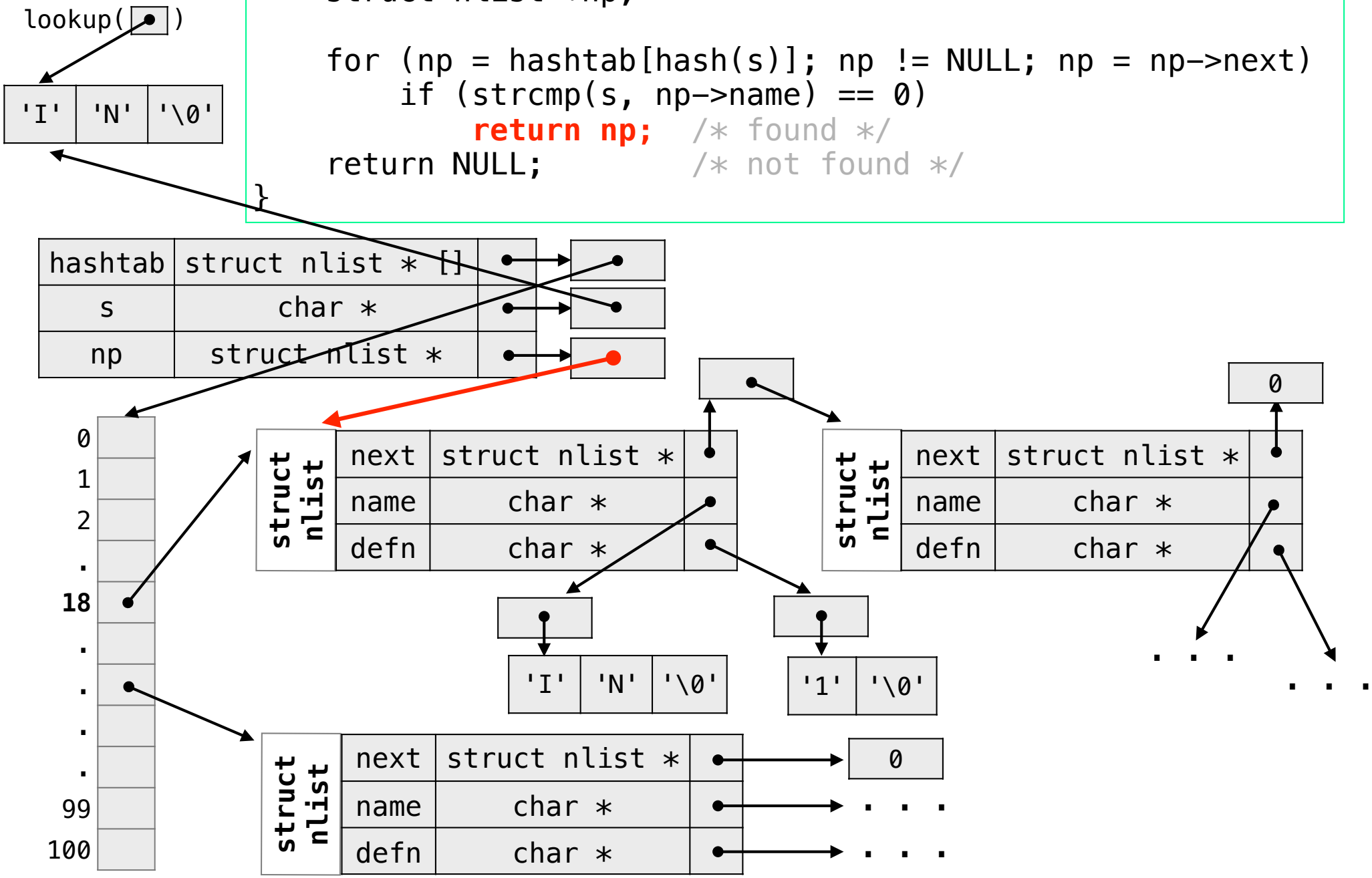
찾기

```
/* lookup: look for s in hashtab */
struct nlist *lookup(char *s) {
    struct nlist *np;

    for (np = hashtab[hash(s)]; np != NULL; np = np->next)
        if (strcmp(s, np->name) == 0)
            return np;  /* found */
    return NULL;        /* not found */
}
```

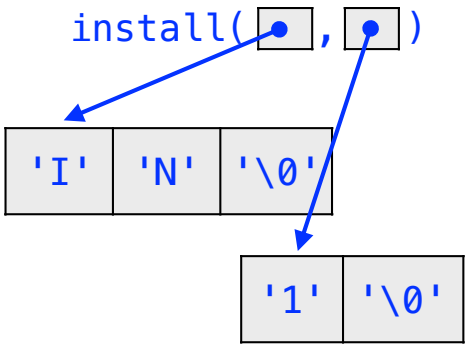


찾기

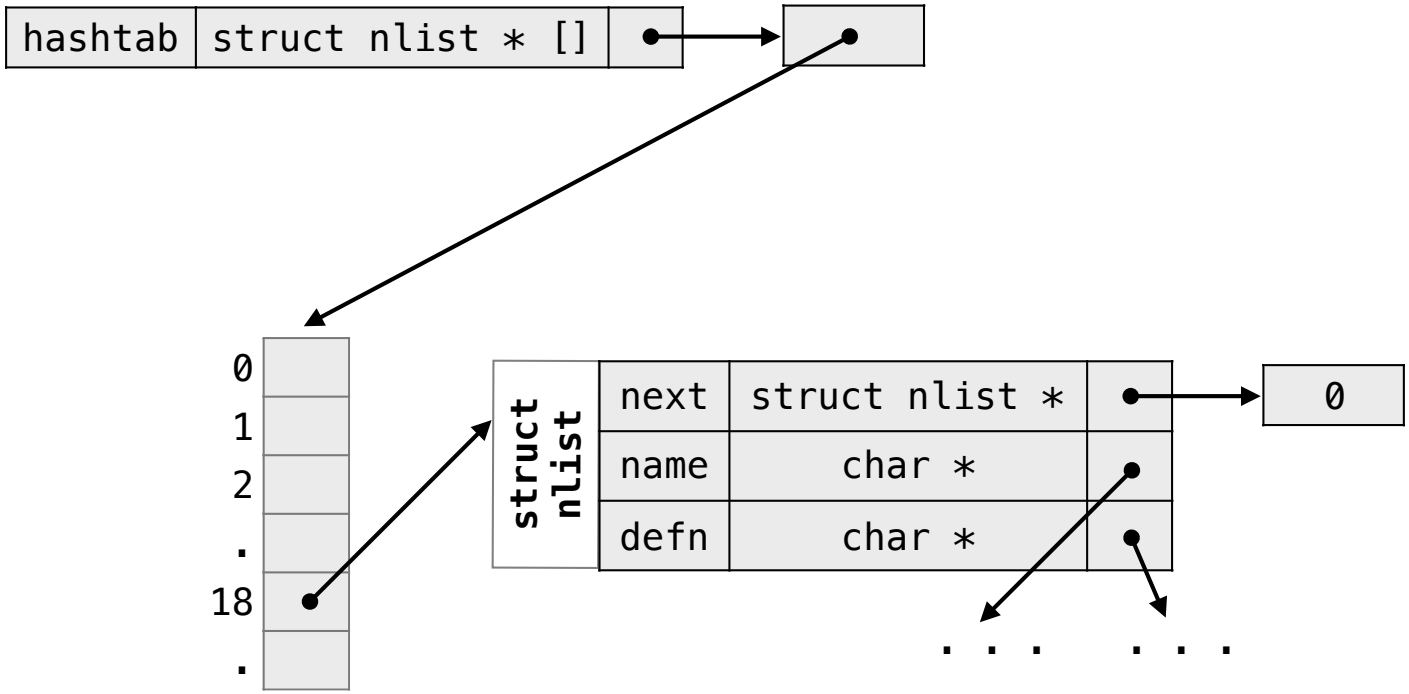
```
/* lookup: look for s in hashtab */
struct nlist *lookup(char *s) {
    struct nlist *np;

    for (np = hashtab[hash(s)]; np != NULL; np = np->next)
        if (strcmp(s, np->name) == 0)
            return np;   /* found */
    return NULL;         /* not found */
}
```



넣기

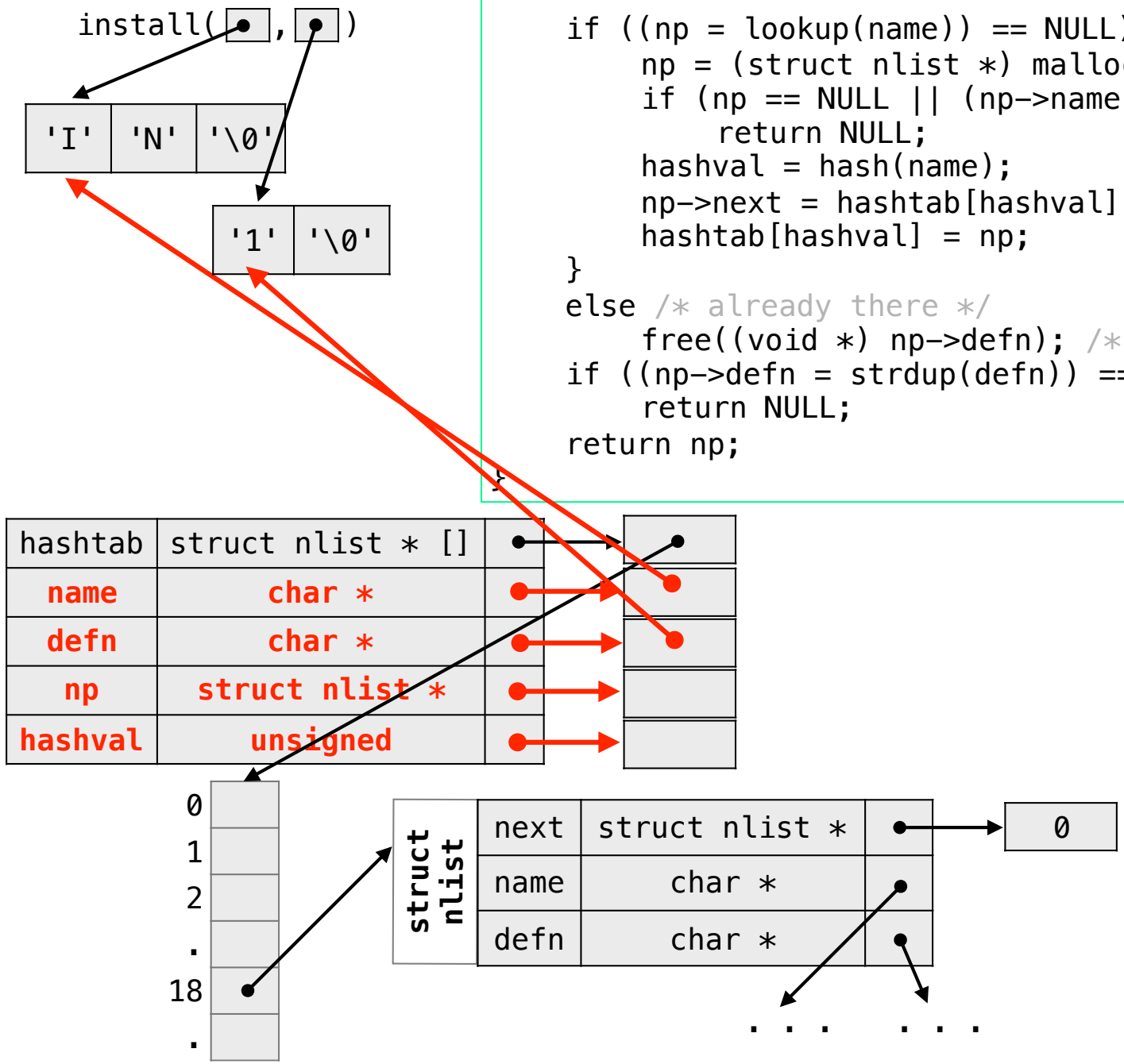



```c
/* install: put (name, defn) in hashtab */
struct nlist *install(char *name, char *defn) {
    struct nlist *np;
    unsigned hashval;

    if ((np = lookup(name)) == NULL) { /* not found */
        np = (struct nlist *) malloc(sizeof(*np));
        if (np == NULL || (np->name = strdup(name)) == NULL)
            return NULL;
        hashval = hash(name);
        np->next = hashtab[hashval];
        hashtab[hashval] = np;
    }
    else /* already there */
        free((void *) np->defn); /* free previous defn */
    if ((np->defn = strdup(defn)) == NULL)
        return NULL;
    return np;
}
```



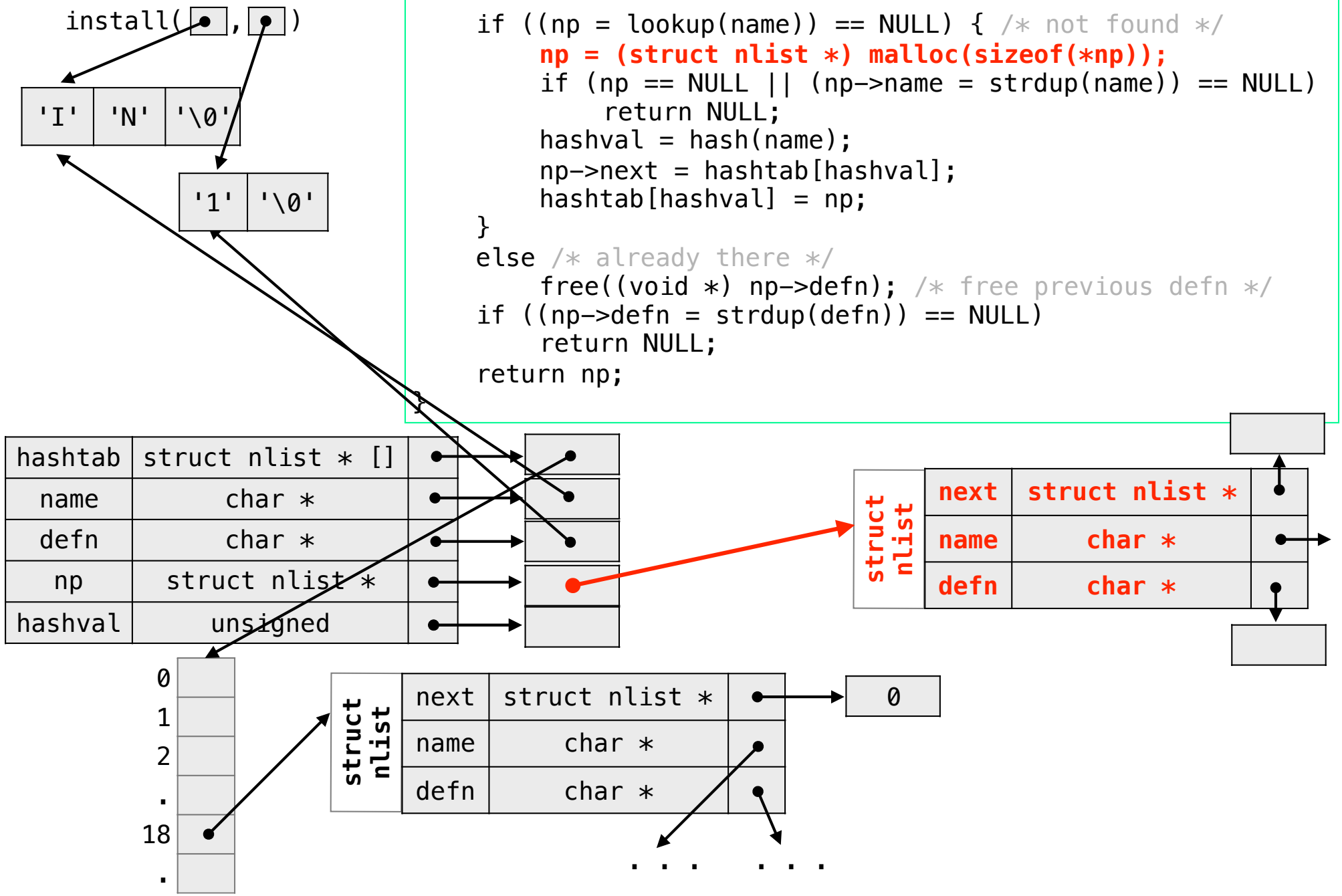
넣기

```
/* install: put (name, defn) in hashtab */
struct nlist *install(char *name, char *defn) {
    struct nlist *np;
    unsigned hashval;

    if ((np = lookup(name)) == NULL) { /* not found */
        np = (struct nlist *) malloc(sizeof(*np));
        if (np == NULL || (np->name = strdup(name)) == NULL)
            return NULL;
        hashval = hash(name);
        np->next = hashtab[hashval];
        hashtab[hashval] = np;
    }
    else /* already there */
        free((void *) np->defn); /* free previous defn */
    if ((np->defn = strdup(defn)) == NULL)
        return NULL;
    return np;
}
```



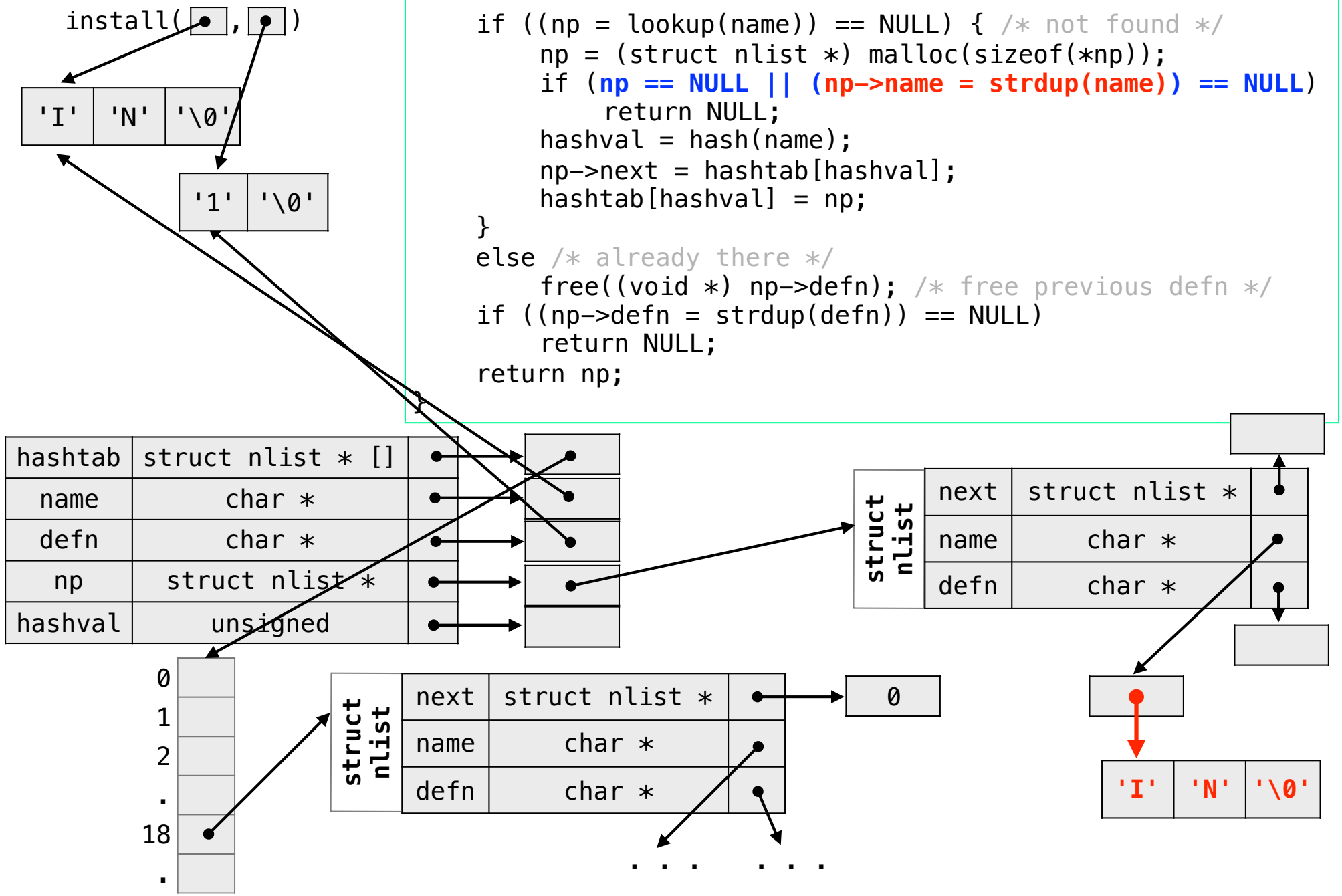
넣기

```
/* install: put (name, defn) in hashtab */
struct nlist *install(char *name, char *defn) {
    struct nlist *np;
    unsigned hashval;

    if ((np = lookup(name)) == NULL) { /* not found */
        np = (struct nlist *) malloc(sizeof(*np));
        if (np == NULL || (np->name = strdup(name)) == NULL)
            return NULL;
        hashval = hash(name);
        np->next = hashtab[hashval];
        hashtab[hashval] = np;
    }
    else /* already there */
        free((void *) np->defn); /* free previous defn */
    if ((np->defn = strdup(defn)) == NULL)
        return NULL;
    return np;
}
```



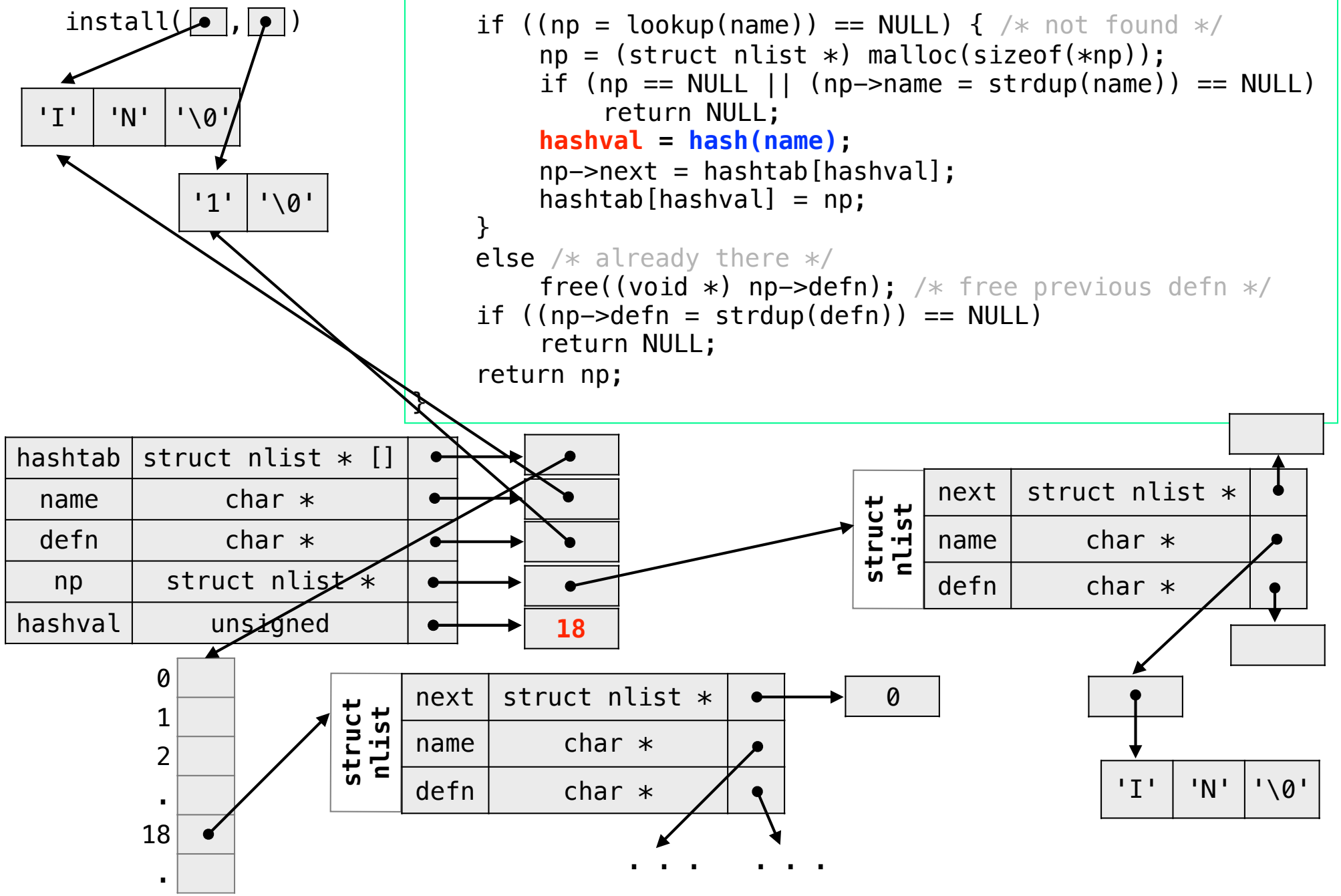
넣기

```
/* install: put (name, defn) in hashtab */
struct nlist *install(char *name, char *defn) {
    struct nlist *np;
    unsigned hashval;

    if ((np = lookup(name)) == NULL) { /* not found */
        np = (struct nlist *) malloc(sizeof(*np));
        if (np == NULL || (np->name = strdup(name)) == NULL)
            return NULL;
        hashval = hash(name);
        np->next = hashtab[hashval];
        hashtab[hashval] = np;
    }
    else /* already there */
        free((void *) np->defn); /* free previous defn */
    if ((np->defn = strdup(defn)) == NULL)
        return NULL;
    return np;
}
```



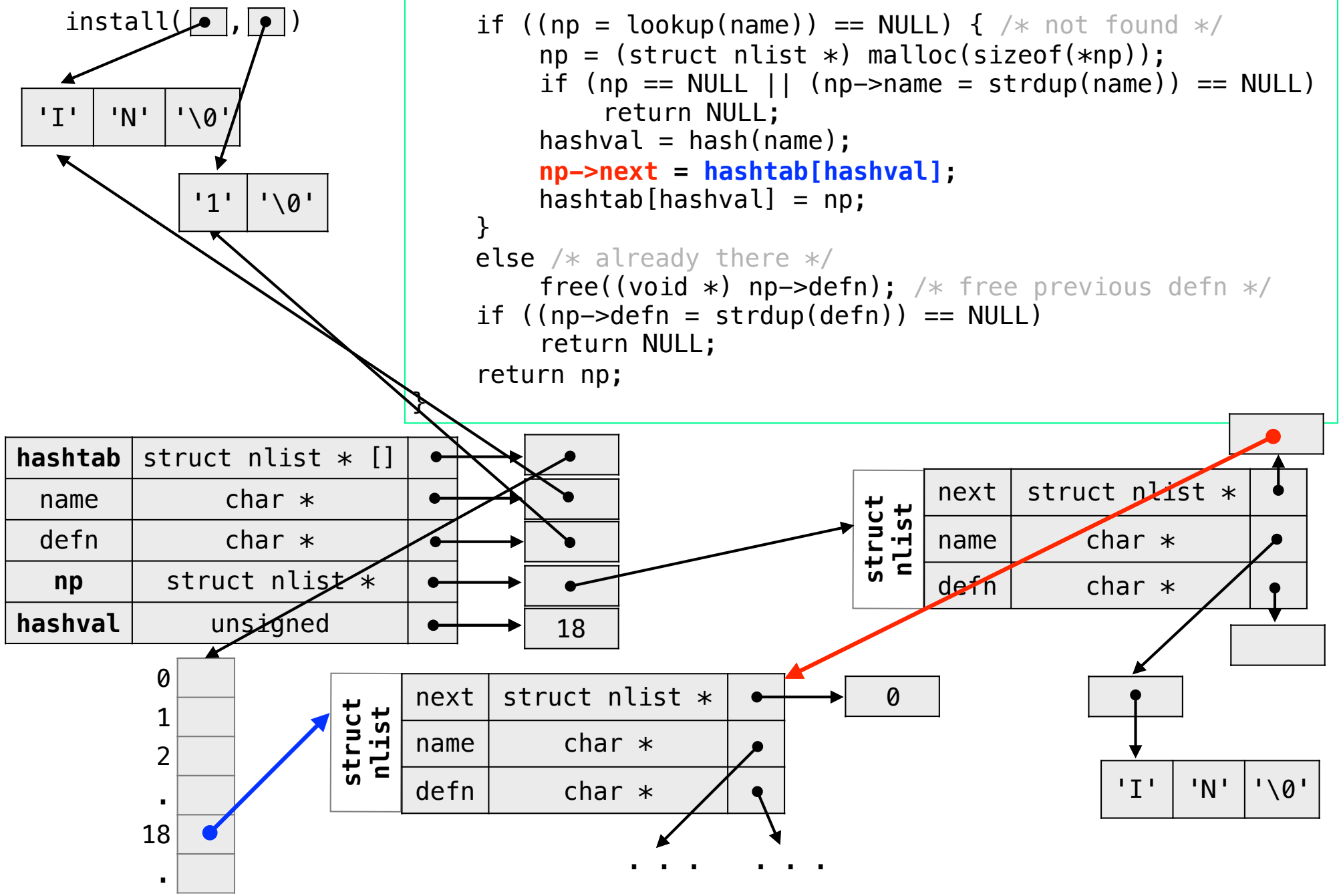
넣기

```
/* install: put (name, defn) in hashtab */
struct nlist *install(char *name, char *defn) {
    struct nlist *np;
    unsigned hashval;

    if ((np = lookup(name)) == NULL) { /* not found */
        np = (struct nlist *) malloc(sizeof(*np));
        if (np == NULL || (np->name = strdup(name)) == NULL)
            return NULL;
        hashval = hash(name);
        np->next = hashtab[hashval];
        hashtab[hashval] = np;
    }
    else /* already there */
        free((void *) np->defn); /* free previous defn */
    if ((np->defn = strdup(defn)) == NULL)
        return NULL;
    return np;
}
```



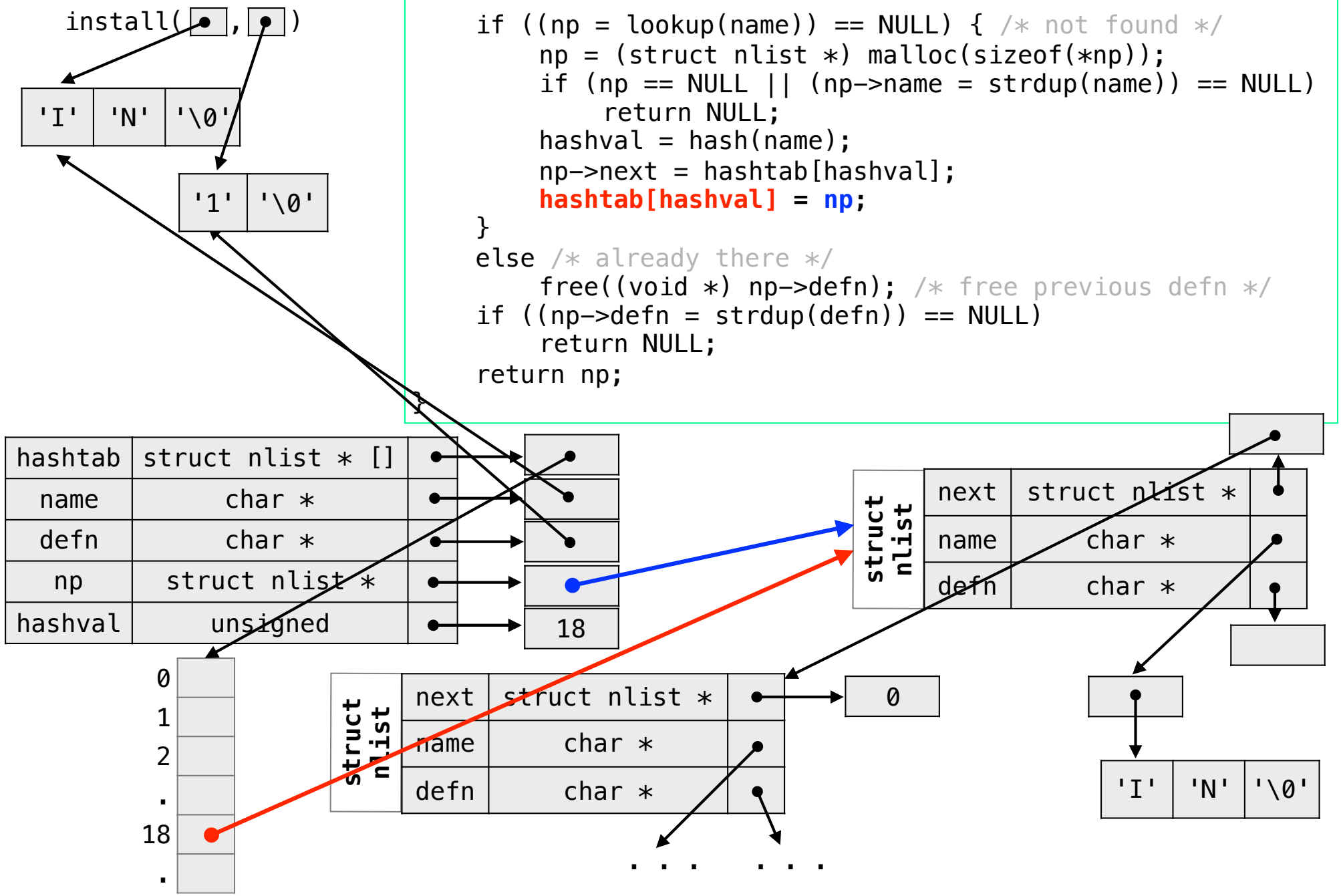
넣기

```
/* install: put (name, defn) in hashtab */
struct nlist *install(char *name, char *defn) {
    struct nlist *np;
    unsigned hashval;

    if ((np = lookup(name)) == NULL) { /* not found */
        np = (struct nlist *) malloc(sizeof(*np));
        if (np == NULL || (np->name = strdup(name)) == NULL)
            return NULL;
        hashval = hash(name);
        np->next = hashtab[hashval];
        hashtab[hashval] = np;
    }
    else /* already there */
        free((void *) np->defn); /* free previous defn */
    if ((np->defn = strdup(defn)) == NULL)
        return NULL;
    return np;
}
```



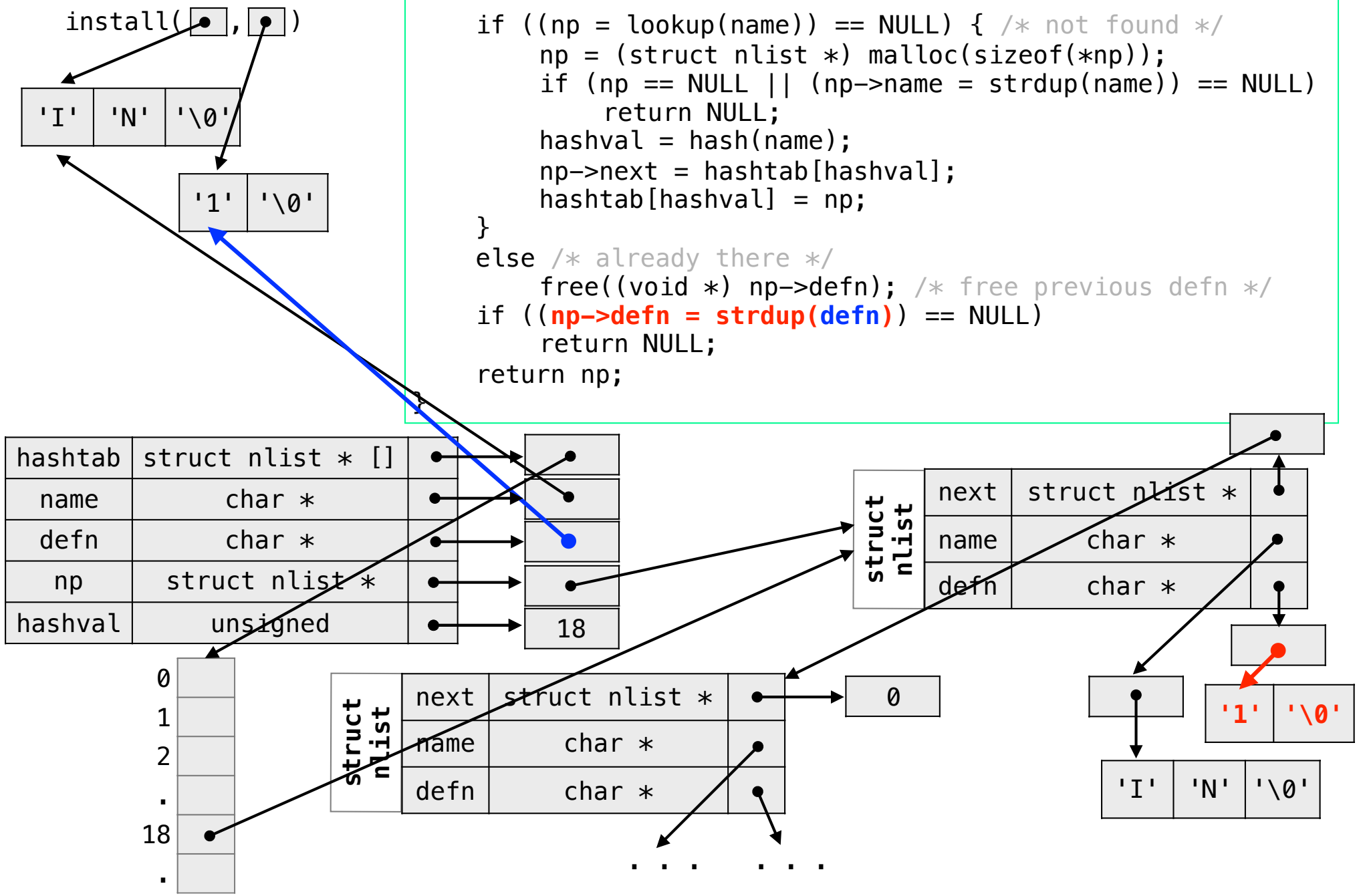
넣기

```
/* install: put (name, defn) in hashtab */
struct nlist *install(char *name, char *defn) {
    struct nlist *np;
    unsigned hashval;

    if ((np = lookup(name)) == NULL) { /* not found */
        np = (struct nlist *) malloc(sizeof(*np));
        if (np == NULL || (np->name = strdup(name)) == NULL)
            return NULL;
        hashval = hash(name);
        np->next = hashtab[hashval];
        hashtab[hashval] = np;
    }
    else /* already there */
        free((void *) np->defn); /* free previous defn */
    if ((np->defn = strdup(defn)) == NULL)
        return NULL;
    return np;
}
```



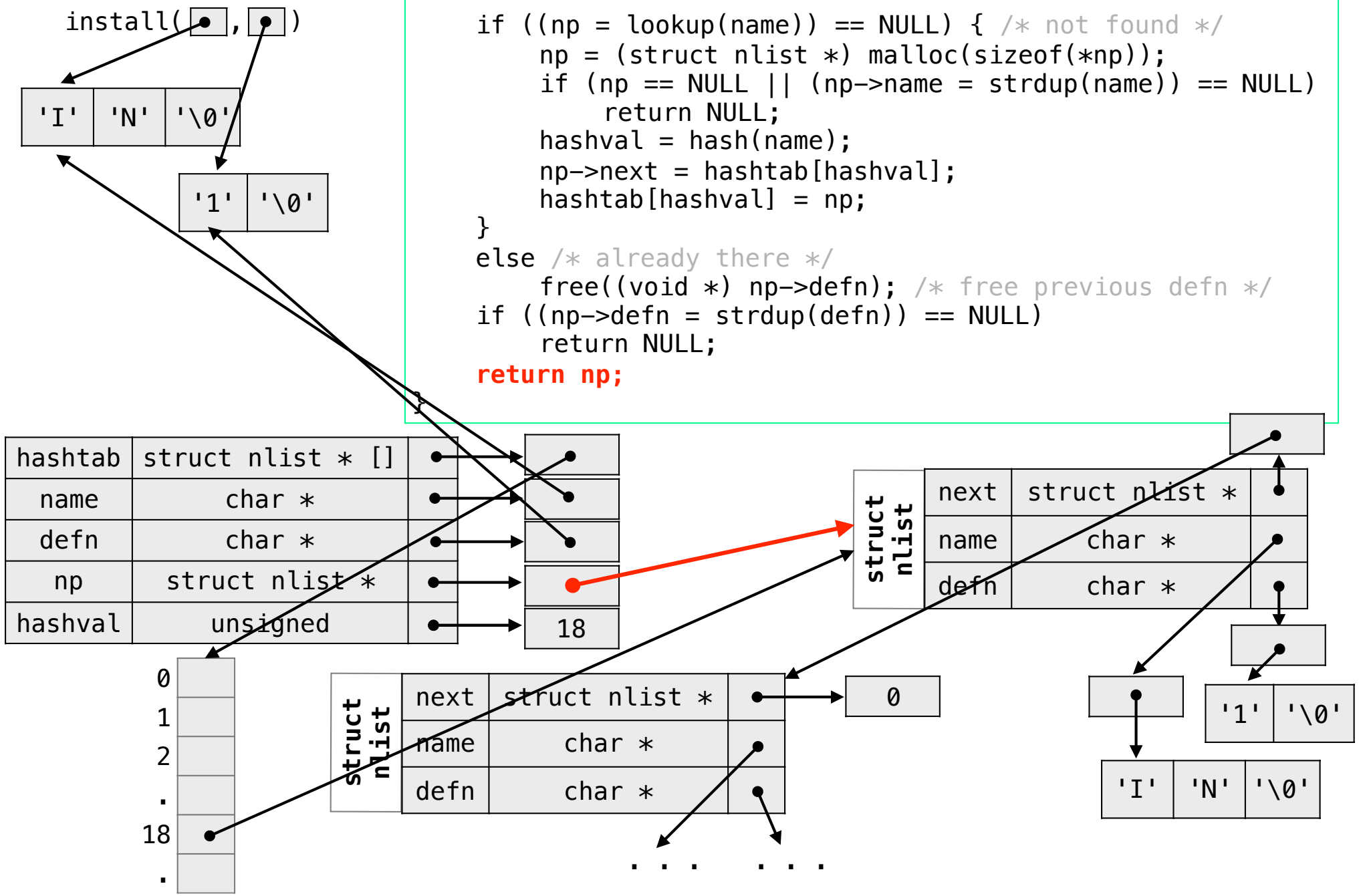
넣기

```
/* install: put (name, defn) in hashtab */
struct nlist *install(char *name, char *defn) {
    struct nlist *np;
    unsigned hashval;

    if ((np = lookup(name)) == NULL) { /* not found */
        np = (struct nlist *) malloc(sizeof(*np));
        if (np == NULL || (np->name = strdup(name)) == NULL)
            return NULL;
        hashval = hash(name);
        np->next = hashtab[hashval];
        hashtab[hashval] = np;
    }
    else /* already there */
        free((void *) np->defn); /* free previous defn */
    if ((np->defn = strdup(defn)) == NULL)
        return NULL;
    return np;
}
```



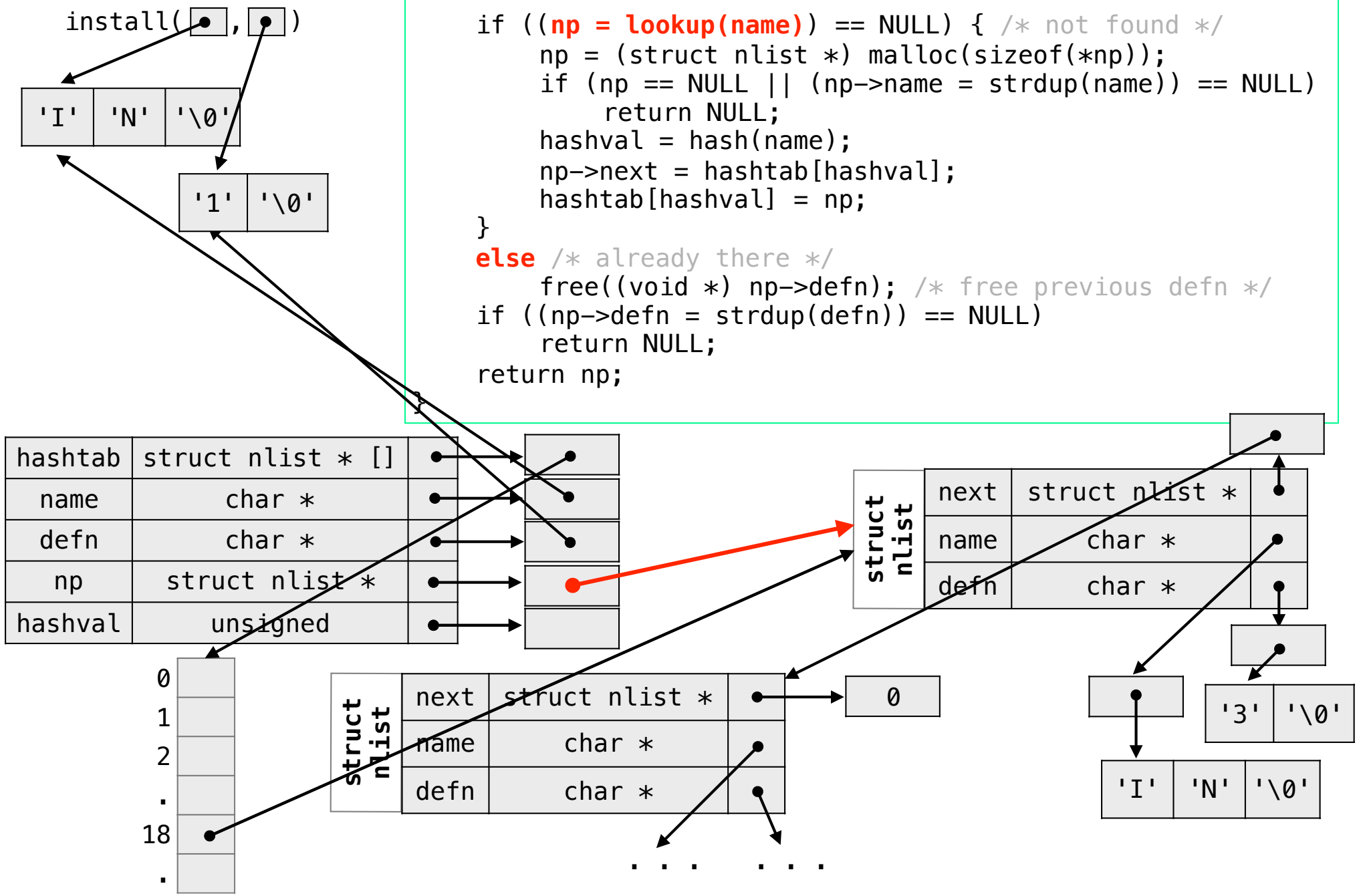
넣기

```
/* install: put (name, defn) in hashtab */
struct nlist *install(char *name, char *defn) {
    struct nlist *np;
    unsigned hashval;

    if ((np = lookup(name)) == NULL) { /* not found */
        np = (struct nlist *) malloc(sizeof(*np));
        if (np == NULL || (np->name = strdup(name)) == NULL)
            return NULL;
        hashval = hash(name);
        np->next = hashtab[hashval];
        hashtab[hashval] = np;
    }
    else /* already there */
        free((void *) np->defn); /* free previous defn */
    if ((np->defn = strdup(defn)) == NULL)
        return NULL;
    return np;
}
```



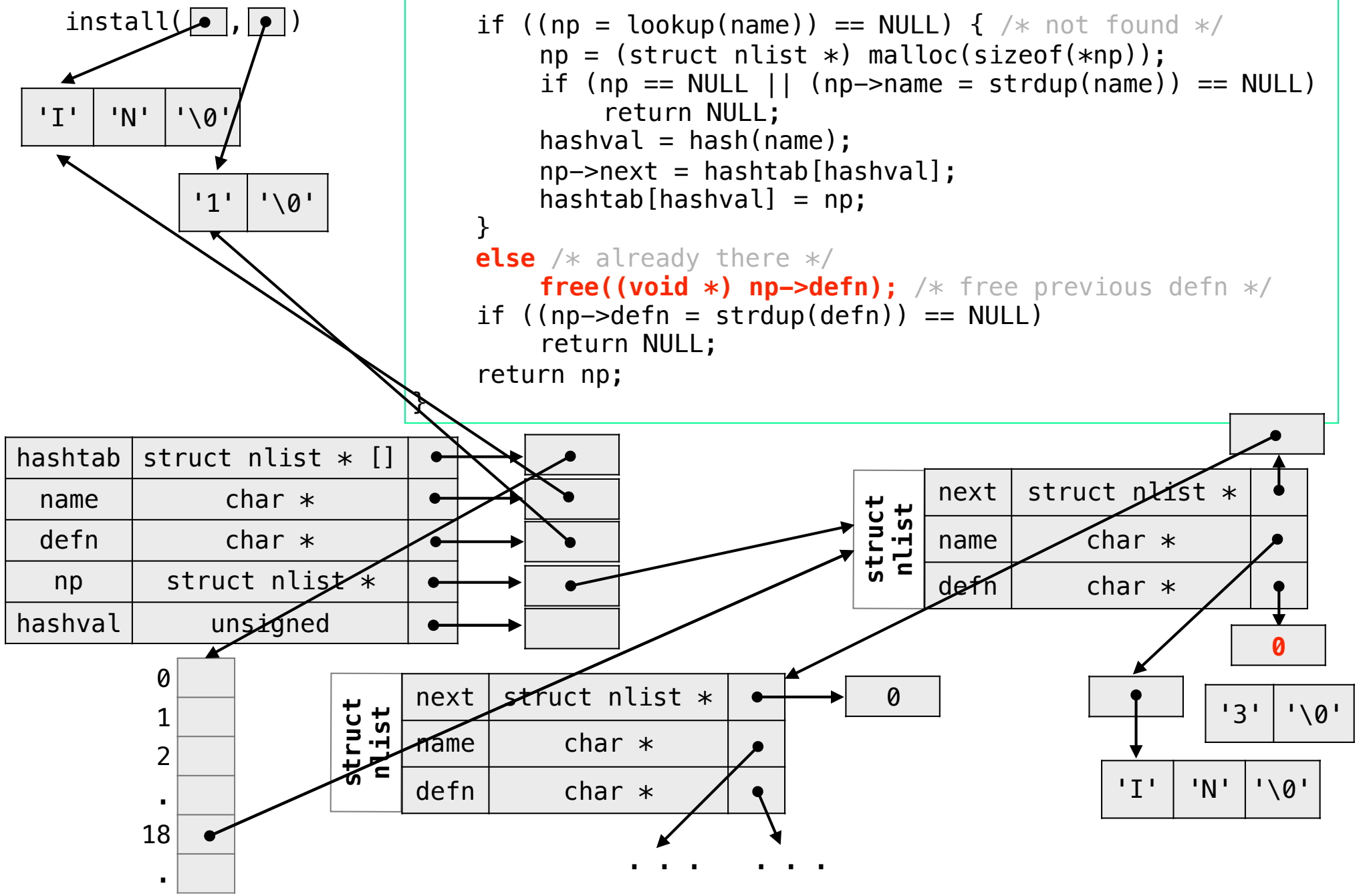
넣기

```
/* install: put (name, defn) in hashtab */
struct nlist *install(char *name, char *defn) {
    struct nlist *np;
    unsigned hashval;

    if ((np = lookup(name)) == NULL) { /* not found */
        np = (struct nlist *) malloc(sizeof(*np));
        if (np == NULL || (np->name = strdup(name)) == NULL)
            return NULL;
        hashval = hash(name);
        np->next = hashtab[hashval];
        hashtab[hashval] = np;
    }
    else /* already there */
        free((void *) np->defn); /* free previous defn */
    if ((np->defn = strdup(defn)) == NULL)
        return NULL;
    return np;
}
```



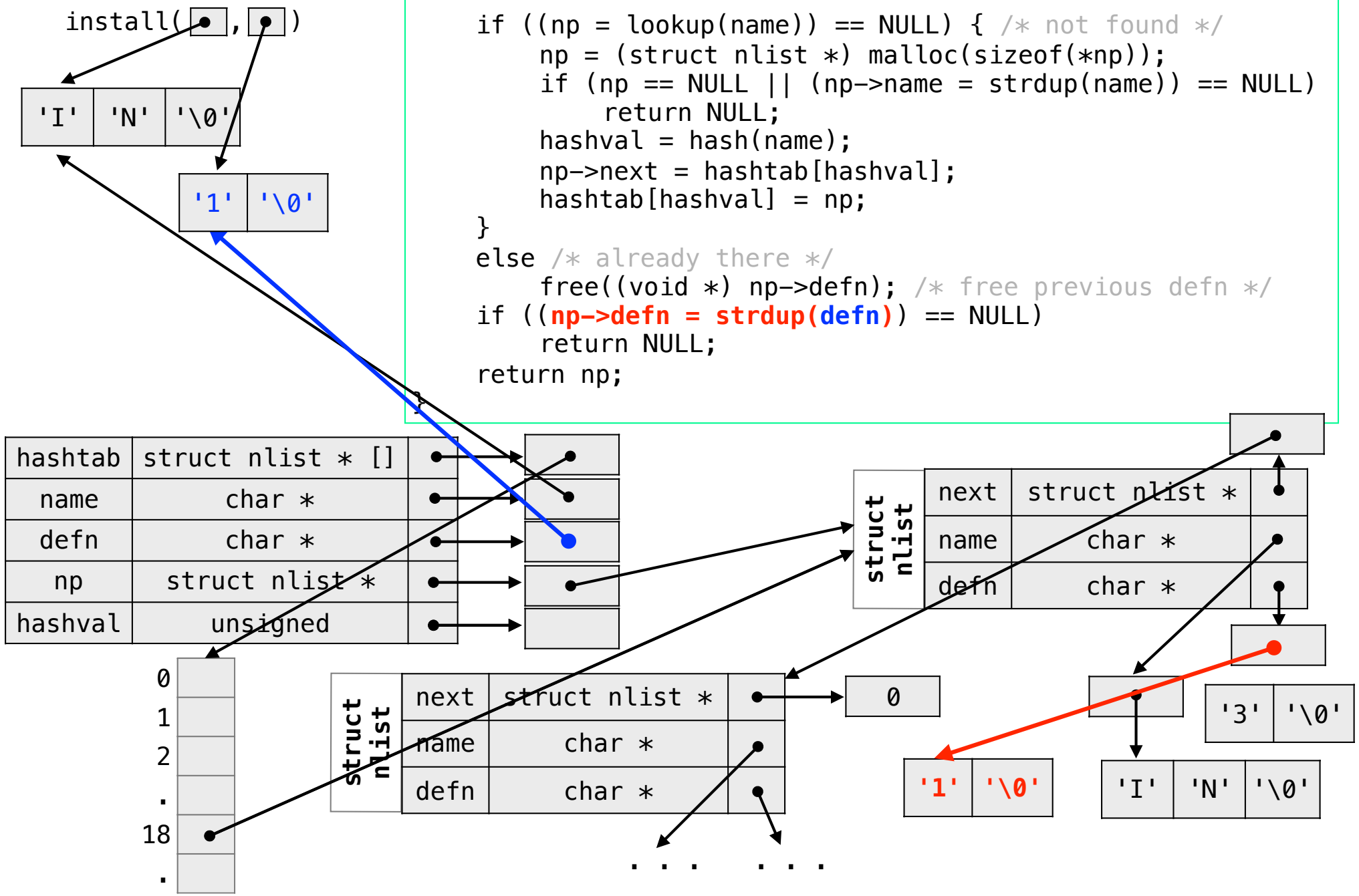
넣기

```
/* install: put (name, defn) in hashtab */
struct nlist *install(char *name, char *defn) {
    struct nlist *np;
    unsigned hashval;

    if ((np = lookup(name)) == NULL) { /* not found */
        np = (struct nlist *) malloc(sizeof(*np));
        if (np == NULL || (np->name = strdup(name)) == NULL)
            return NULL;
        hashval = hash(name);
        np->next = hashtab[hashval];
        hashtab[hashval] = np;
    }
    else /* already there */
        free((void *) np->defn); /* free previous defn */
    if ((np->defn = strdup(defn)) == NULL)
        return NULL;
    return np;
}
```



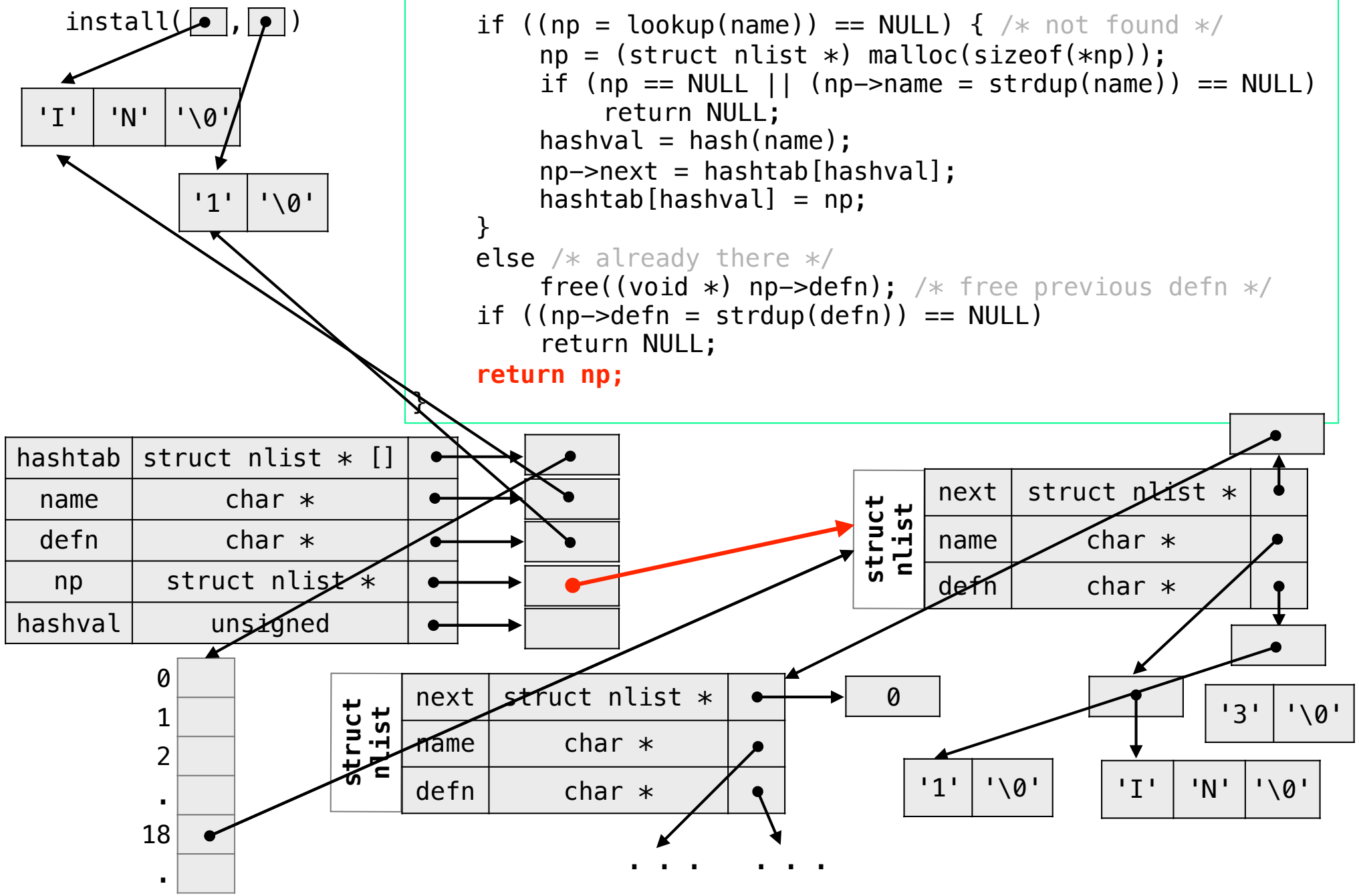
넣기

```
/* install: put (name, defn) in hashtab */
struct nlist *install(char *name, char *defn) {
    struct nlist *np;
    unsigned hashval;

    if ((np = lookup(name)) == NULL) { /* not found */
        np = (struct nlist *) malloc(sizeof(*np));
        if (np == NULL || (np->name = strdup(name)) == NULL)
            return NULL;
        hashval = hash(name);
        np->next = hashtab[hashval];
        hashtab[hashval] = np;
    }
    else /* already there */
        free((void *) np->defn); /* free previous defn */
    if ((np->defn = strdup(defn)) == NULL)
        return NULL;
    return np;
}
```



넣기

```
/* install: put (name, defn) in hashtab */
struct nlist *install(char *name, char *defn) {
    struct nlist *np;
    unsigned hashval;

    if ((np = lookup(name)) == NULL) { /* not found */
        np = (struct nlist *) malloc(sizeof(*np));
        if (np == NULL || (np->name = strdup(name)) == NULL)
            return NULL;
        hashval = hash(name);
        np->next = hashtab[hashval];
        hashtab[hashval] = np;
    }
    else /* already there */
        free((void *) np->defn); /* free previous defn */
    if ((np->defn = strdup(defn)) == NULL)
        return NULL;
    return np;
}
```



Typedef

타입의 이명(동의어, 별명) 만드는 기능

```
typedef int Length;          => Length는 int의 이명

Length len, maxlen;
Length *lengths[];
```

```
typedef char *String;        => String는 char *의 이명

String p, lineptr[MAXLINES], alloc(int);
int strcmp(String, String);
p = (String) malloc(100);
```

```
typedef struct tnode *Treeptr;

typedef struct tnode {
    char *word;
    int count;
    TreePtr left;
    TreePtr right;
} Treenode;

Treeptr talloc(void) {
    return (Treeptr) malloc(sizeof(Treenode));
}
```

```
typedef int (*PFI)(char *, char *);

PFI strcmp, numcmp;
```

Typedef

이 기능의 존재 가치는?

이식성

- 하드웨어 종속성이 있는 타입의 경우, 다른 하드웨어에서 프로그램을 작동시키려 할때 `typedef` 만 수정하면 간편하게 문제 해결
- `short`, `int`, `long` 하드웨어 별로 이명 지정
- 표준 라이브러리 타입 : `size_t`, `ptrdiff_t`

가독성

- 사례: `Treenode`, `Treeptr`

Union

- 상황에 따라 다른 타입 또는 크기의 값을 지닐 수 있는 변수
- 동일 공간에서 다른 종류의 데이터 취급(처리)할 수 있는 기능 제공

```
union u_tag {
    int ival;
    float fval;
    char *sval;
} u;
```

- 변수 u는 int, float, char * 값이면 뭐든지 저장할 수 있는만큼 충분한 공간을 갖고 있어야 함
- 변수 u의 타입은 가장 최근에 저장된 값의 타입으로 결정
- 실행 중, 타입의 일관성을 지키도록 하는 건 프로그래머의 책임

접근 방법

union-name.member

union-name->member

```
if (utype == INT)
    printf("%d\n", u.ival);
else if (utype == FLOAT)
    printf("%f\n", u.fval);
else if (utype ==  STRING)
    printf("%s\n", u.sval);
else
    printf("bad type %d in utype\n", type);
```

Union

```
struct {
    char *name;
    int flags;
    int utype;
    union {
        int ival;
        float fval;
        char *sval;
    } u;
} symtab[NSYM];
```

`symtab[i].u.ival`	멤버 `ival` 정수값
`*symtab[i].u.sval` `symtab[i].u.sval[0]`	멤버 `sval` 문자열의 첫 문자

Bit operations

```
#define KEYWORD  01
#define EXTERNAL 02
#define STATIC   04
```

```
enum { KEYWORD = 01, EXTERNAL = 02, STATIC = 04 };
```

0	0	1

0	1	0

1	0	0

---

flags

?	?	?	?	?	?	?	?

```
flags |= (EXTERNAL | STATIC);
```

flags

?	?	?	?	?	1	1	?

---

flags

?	?	?	?	?	?	?	?

```
flags &= ~(EXTERNAL | STATIC);
```

flags

?	?	?	?	?	0	0	?

---

flags

?	?	?	?	?	1	1	?

```
if ((flags & (EXTERNAL | STATIC)) == 0) . . .
```

→ true

Bit-field (Field)

machine dependent

```
struct {
    unsigned int is_keyword : 1;
    unsigned int is_extern : 1;
    unsigned int is_static : 1;
} flags;
```

필드 너비

- 변수 flags는 3개의 1비트 짜리 필드(멤버)로 구성

flags	?	?	?	?	?	?	?	?

```
flags.is_extern = flags.is_static = 1;
```

flags	?	?	?	?	?	1	1	?

flags	?	?	?	?	?	?	?	?

```
flags.is_extern = flags.is_static = 0;
```

flags	?	?	?	?	?	0	0	?

flags	?	?	?	?	?	1	1	?

```
if (flags.is_extern == 0 && flags.is_static == 0) . . .
```

⟼ true