# [B6]
# リスト処理

# アルゴリズム ＋ データ構造 ＝ プログラム

- データ構造
  - スタック　stack　(B5)
  - リスト　list　(B6)
  - 二分木　binary tree　(B6)
- アルゴリズム
  - ソート　sort　(B7)
  - 探索　search　(B8)
  - ハッシュ　hash　(B8)

# データ構造「リスト」

- 同じ型の複数データを束ねたデータ構造
  - 配列　：データ個数（長さ）が固定
  - リスト：データ個数を可変できる
- 節点（ノード）をポインタでつなぐ
  - 自分と同じ型を指し示すポインタをメンバに持つ構造体（自己参照構造体）

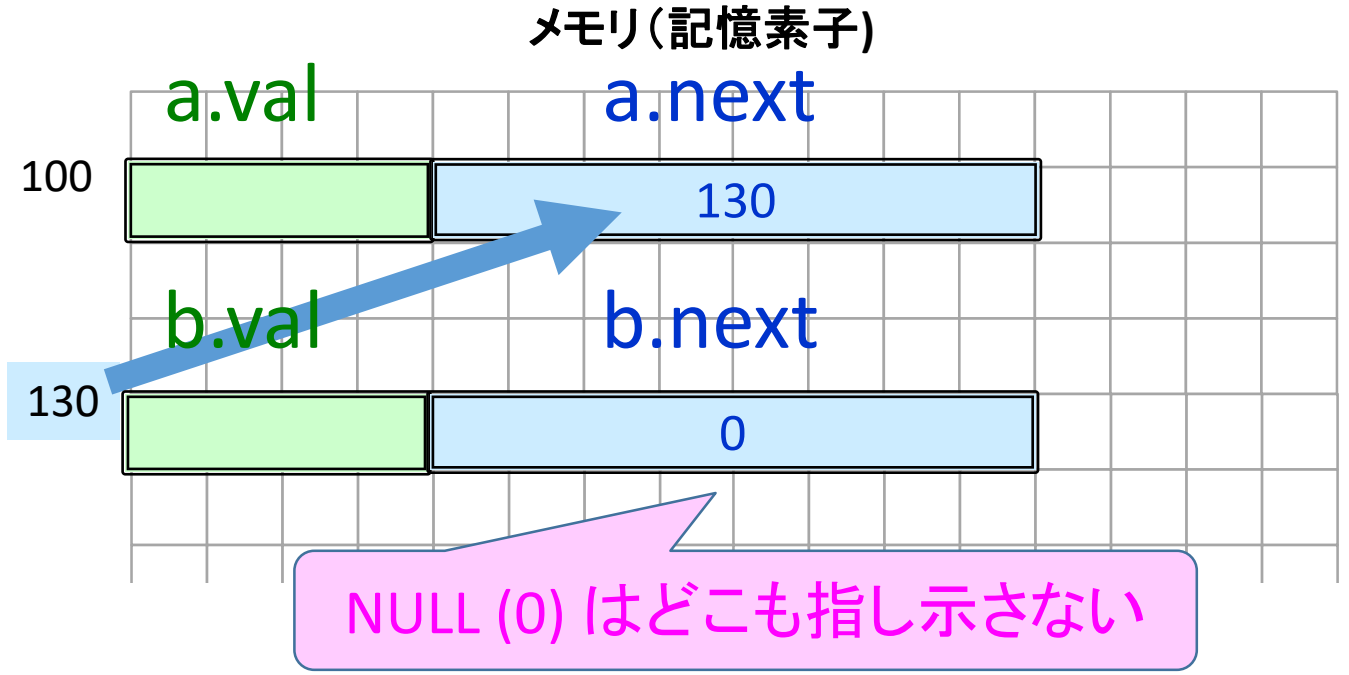
```
typedef struct node_s {
  int              value;
  struct node_s  *next;
} node_t;
```

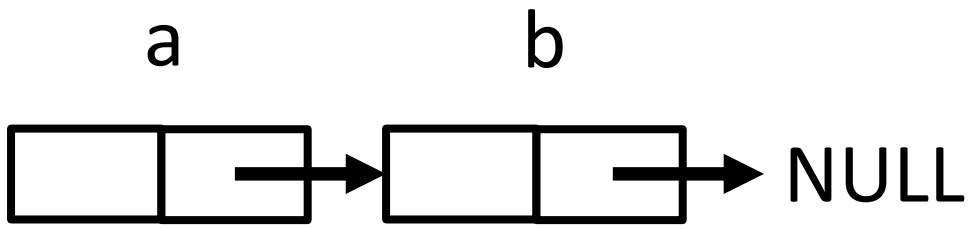
# ポインタと矢印表現

```
typedef struct list_node_s {
  int                    val;
  struct list_node_s  *next;
} list_node_t;

list_node_t a, b;

a.next = &b;
b.next = NULL;
```

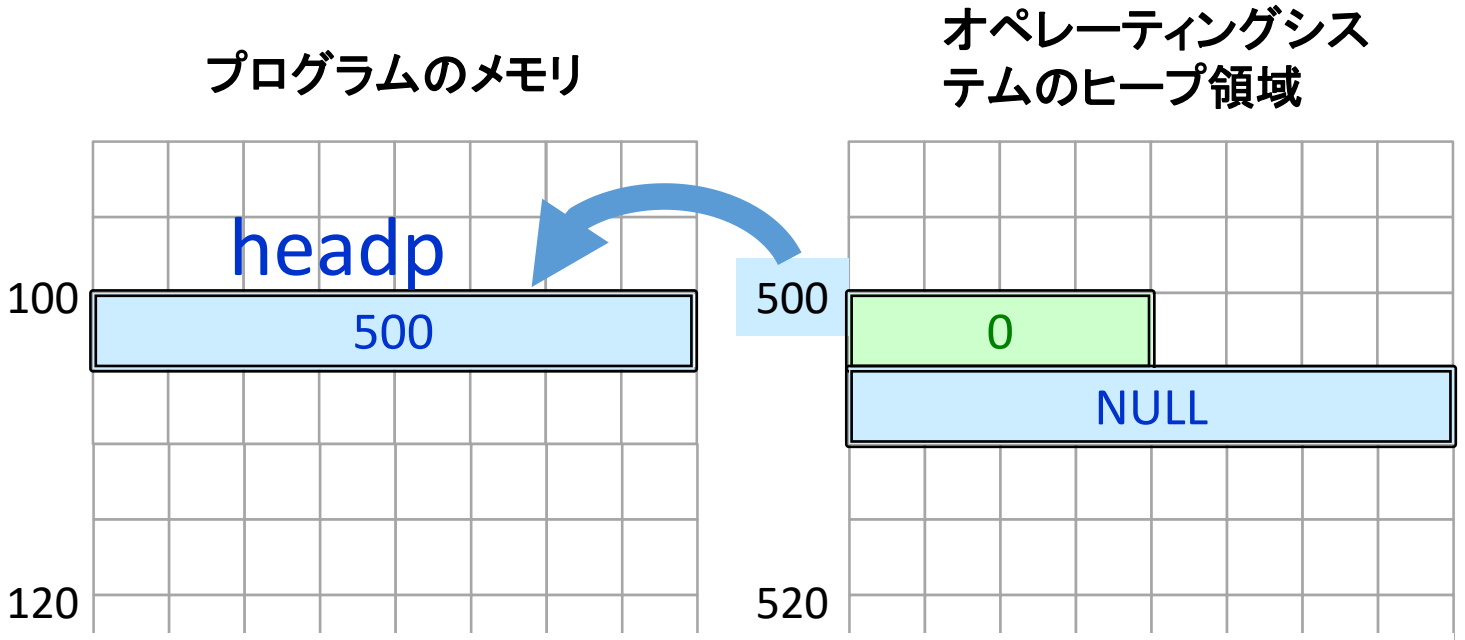


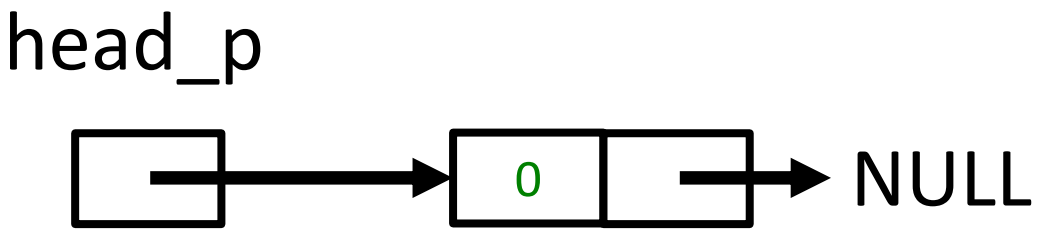




_s は 構造体(structure)
_t は「型」(type)

# create_node()

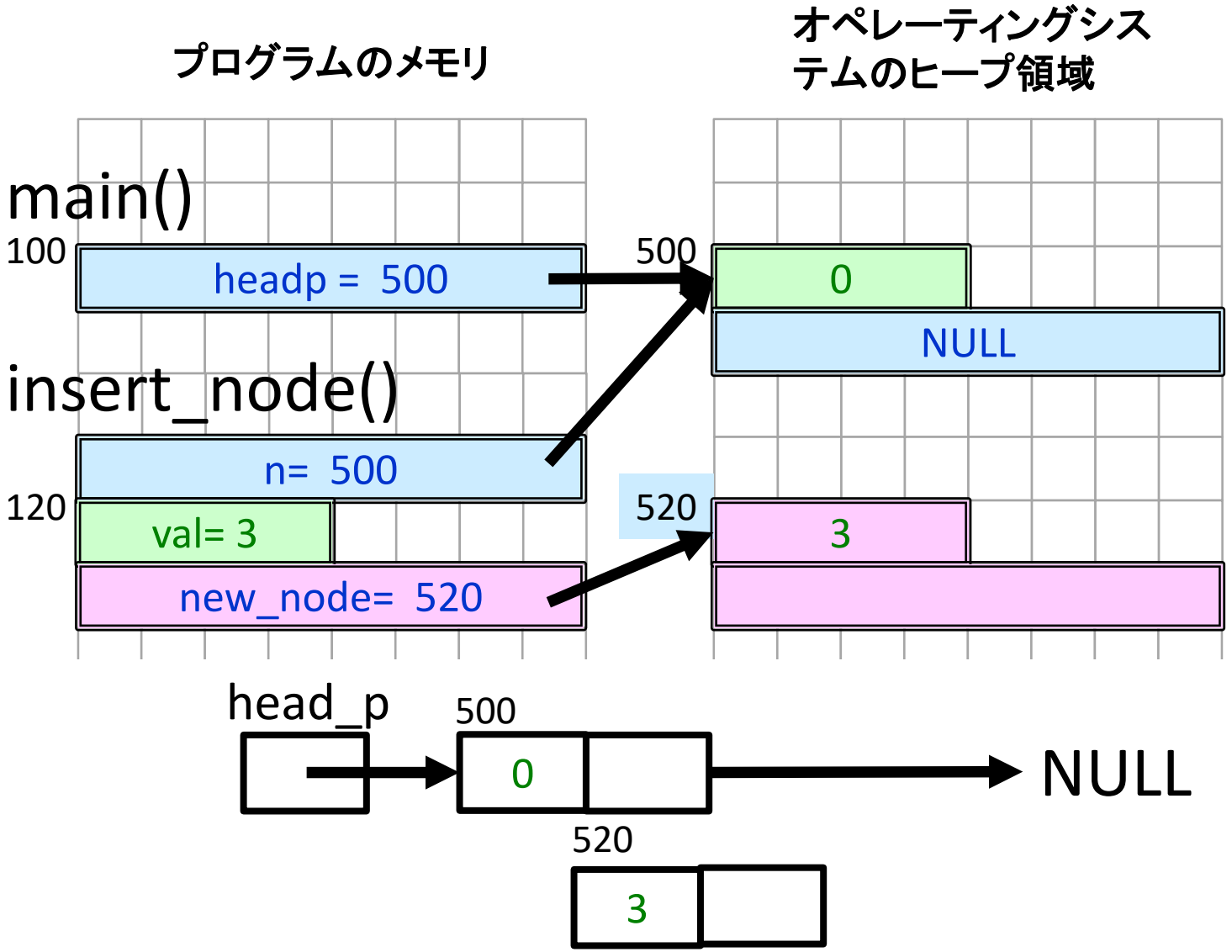
```
#include <stdio.h>
#include <stdlib.h>
#include  “list.h”
int main(void)
{
  list_node_t  *headp;
  headp = create_node(0);
 …
}
```



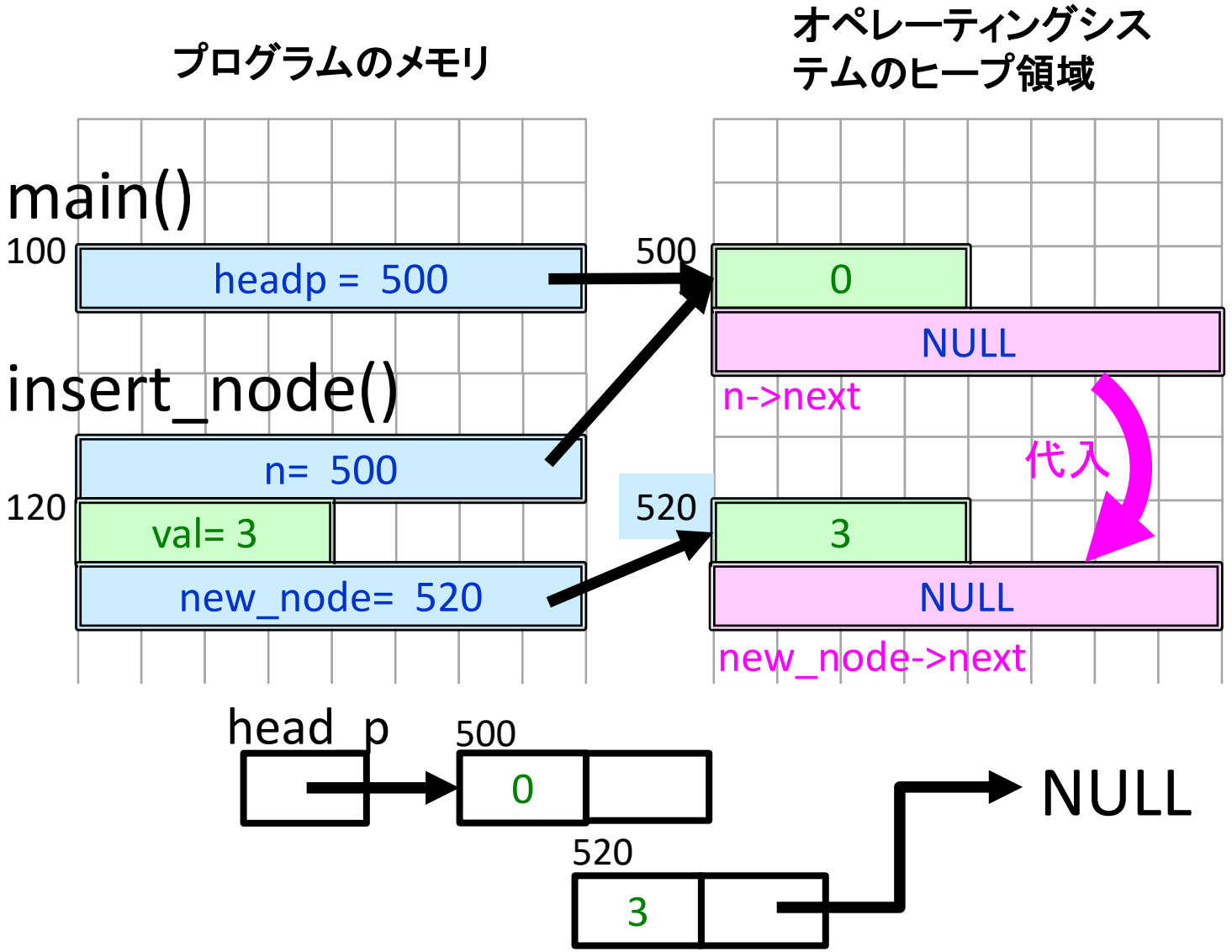
# insert_node()（その1）

```
int main(void)
{
  list_insert(headp, 3);
}
list_insert(…) {
  p = insert_node(head_p,val);
  …
}
insert_node(list_node_t *n,…) {
  list_node_t  *new_node;
  new_node = create_node(val);
  new_node->next = n->next;
  n->next = new_node;
}
```



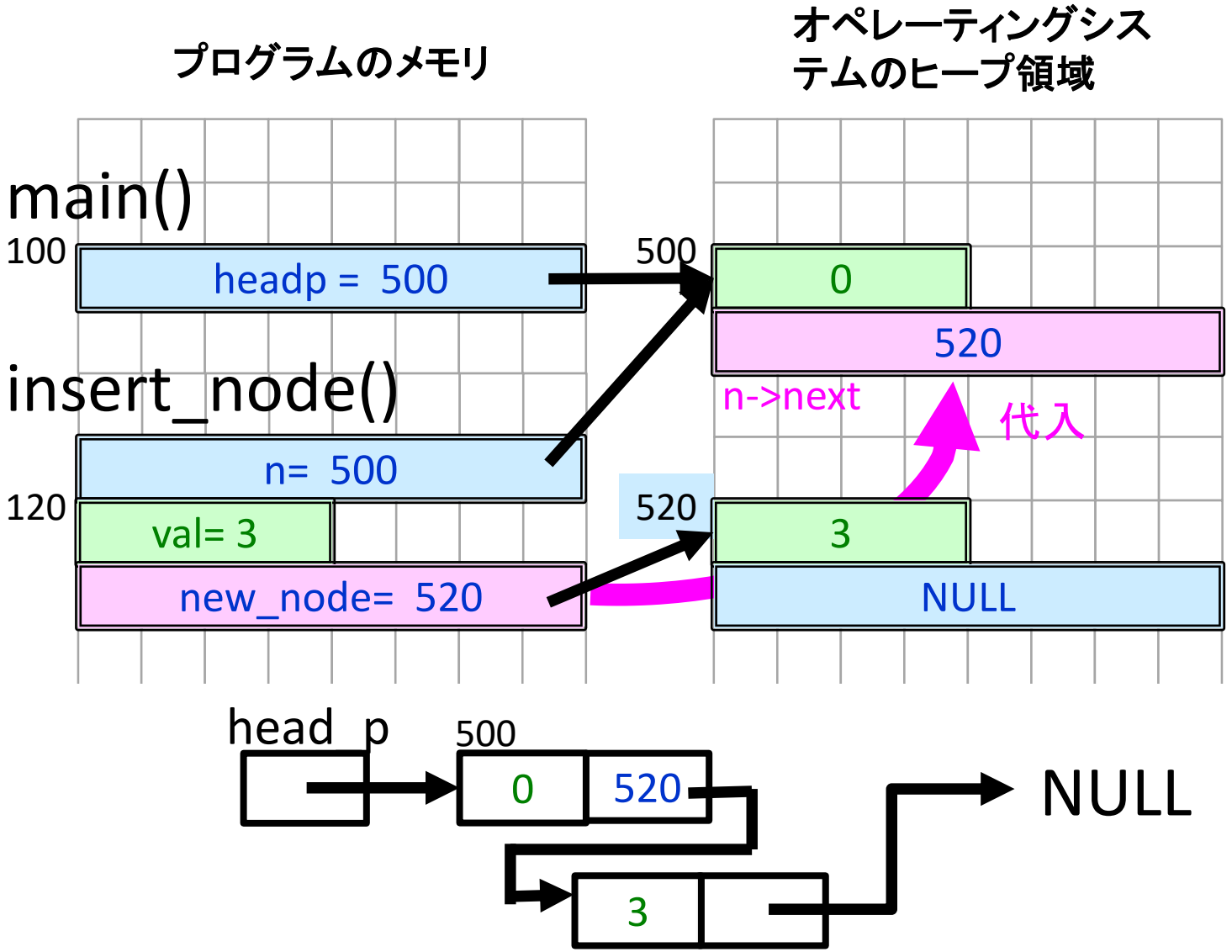
# insert_node()（その2）

```
int main(void)
{
  list_insert(headp, 3);
}
list_insert(…) {
  p = insert_node(head_p,val);
  …
}
insert_node(list_node_t *n,…) {
  list_node_t  *new_node;
  new_node = create_node(val);
  new_node->next = n->next;
  n->next = new_node;
}
```



# insert_node()（その3）

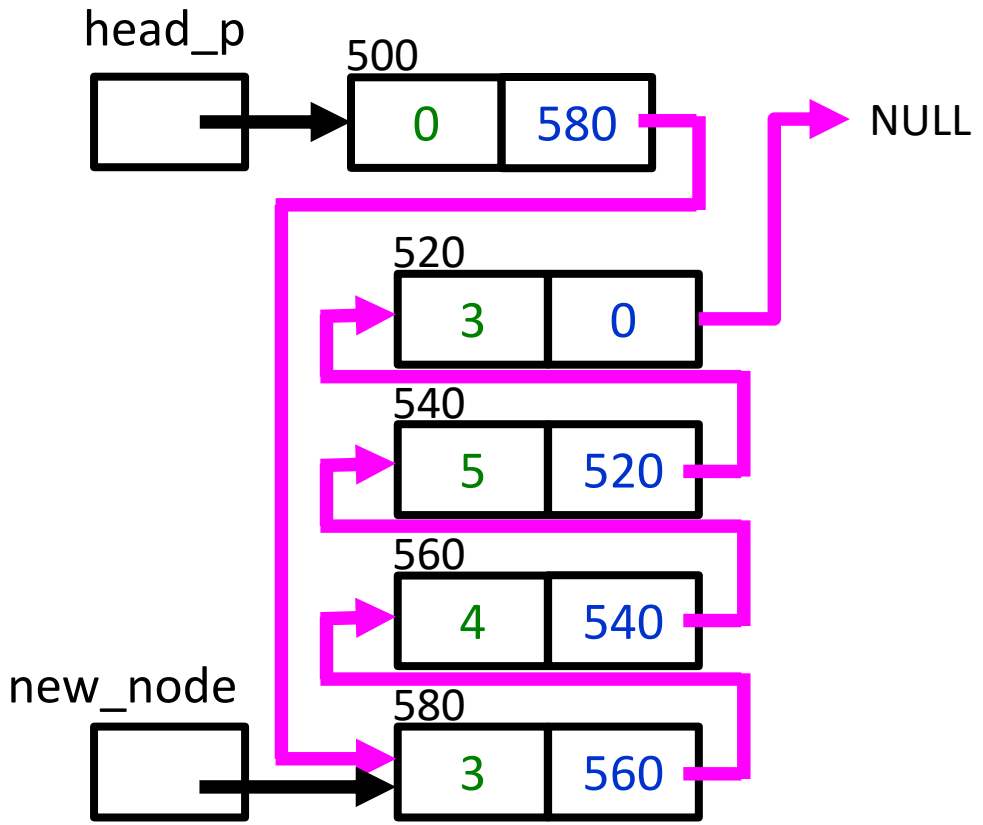
```
int main(void)
{
  list_insert(headp, 3);
}
list_insert(…) {
  p = insert_node(head_p,val);
  …
}
insert_node(list_node_t *n,…) {
  list_node_t  *new_node;
  new_node = create_node(val);
  new_node->next = n->next;
  n->next = new_node;
}
```



# 入力 3, 5, 4, 3 に対するリストの成長

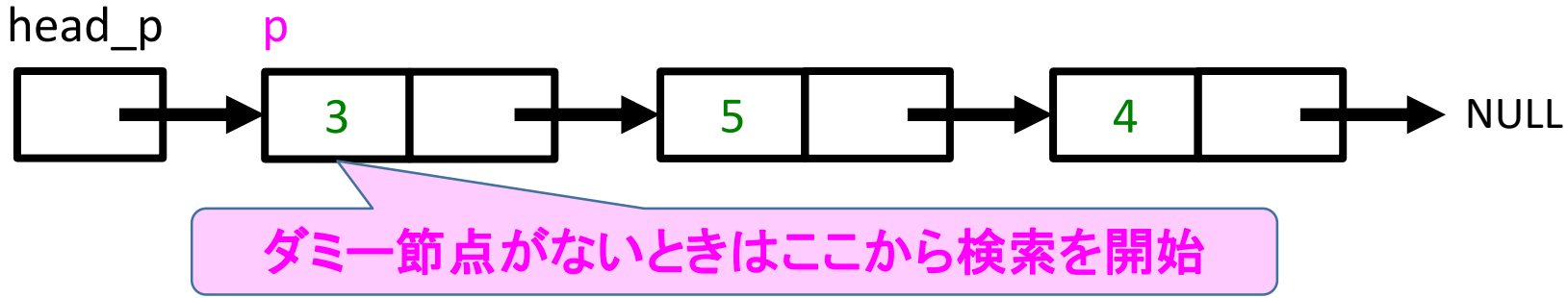
```
#include <stdio.h>
#include <stdlib.h>
#include  “list.h”
int main(void)
{
  list_node_t  *headp;
  headp = create_node(0);

 list_insert(head_p, 3);
 list_insert(head_p, 5);
 list_insert(head_p, 4);
 list_insert(head_p, 3);
 …
}
```



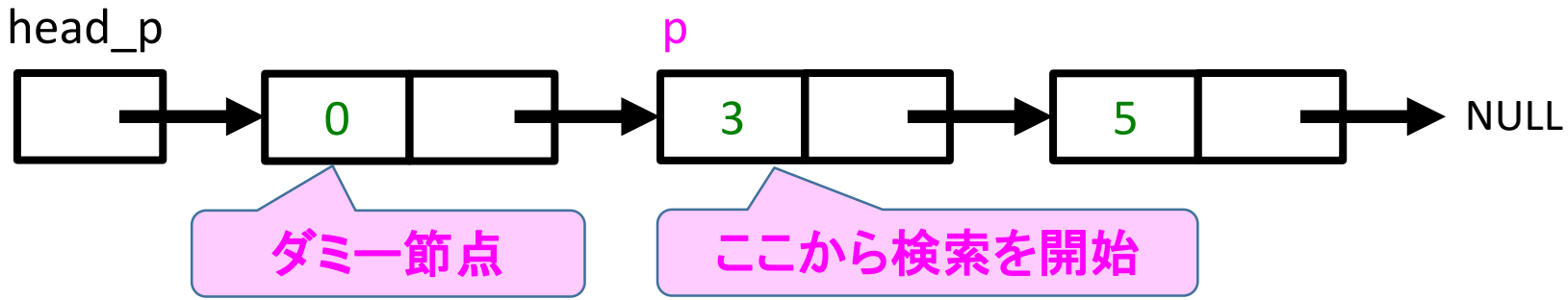
# 節点をたどる for(;;) ループ
## (その1)ダミー節点がない場合

```
#include  “list.h”
int main(void)
{
  list_node_t  *headp, *p;
  headp = create_node(0);
 …
  for (p=head_p; p != NULL ; p = p->next ) {
    if (p->val == search_value) { printf( “I found it!¥n” ); exit(0) }
  }
}
```



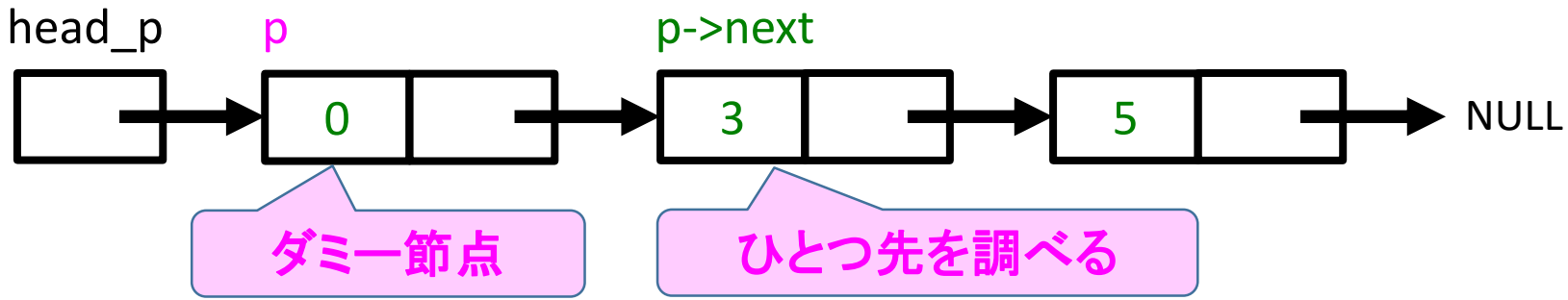
# 節点をたどる for(;;) ループ
## (その2)ダミー節点を使う場合

```
#include  “list.h”
int main(void)
{
  list_node_t  *headp, *p;
  headp = create_node(0);
 …
  for (p=head_p->next; p != NULL ; p = p->next ) {
    if (p->val == search_value) { printf( “I found it!¥n” ); exit(0) }
  }
}
```



# 節点をたどる for(;;) ループ
## (その3)ダミー節点あり、先読み

```
#include  “list.h”
int main(void)
{
  list_node_t  *headp, *p;
  headp = create_node(0);
 …
  for (p=head_p; p->next != NULL ; p = p->next ) {
    if (p->next->val == search_value) { printf( “I found it!¥n” ); exit(0) }
  }
}
```



# データ構造「2分木(にぶんき)」

- 枝分かれする「木(き)」状のデータ構造
- 節点（ノード）をポインタでつなぐ
  - 自分と同じ型を指し示すポインタを「２つ」メンバに持つ構造体
    （自己参照構造体）

```
typedef struct tree_node_s {
  struct tree_node_s *left;
  struct tree_node_s *right;
  int data;
} tree_node_t;
```