C Programming
Day20.B

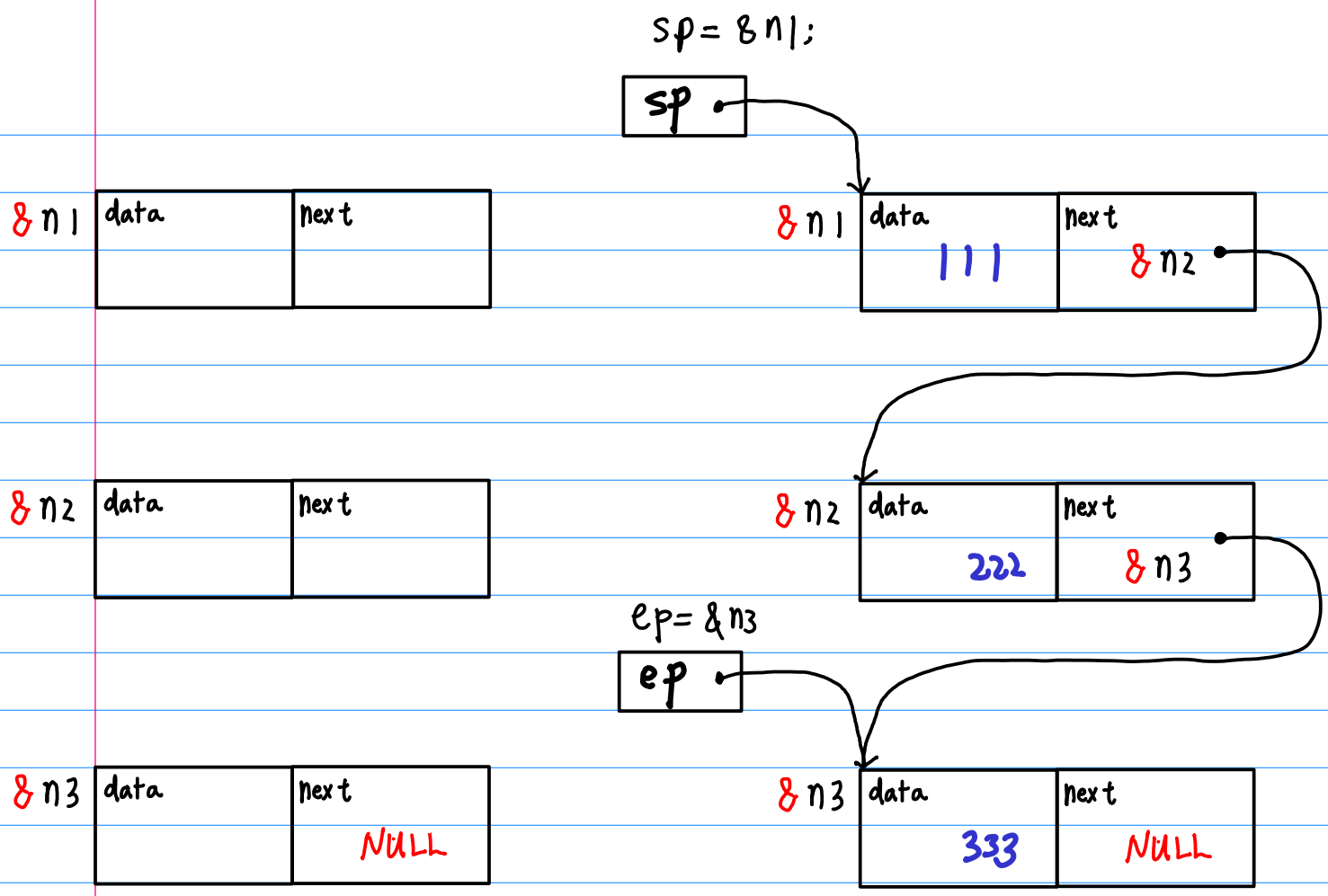
2017.12.01

Introduction to Data Structures

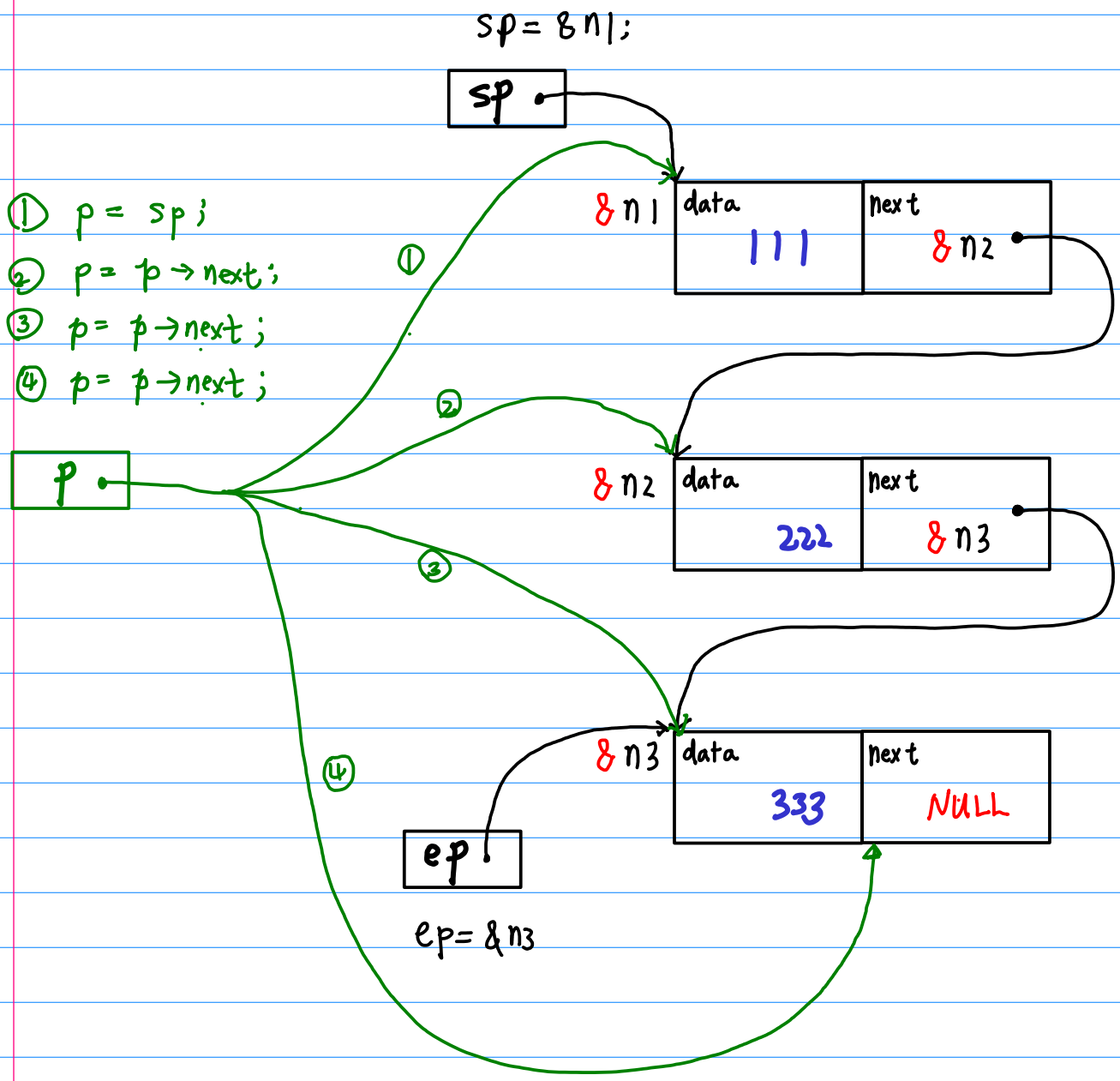
Copyright (c) 2015 - 2017 Young W. Lim.
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

```c
#include <stdio.h>

struct aaa {
  int         data;   // 4-byte
  struct aaa *next;   // 8-byte
} ;


typedef struct node node;

struct node {
  int   data;   // 4-byte
  node *next;   // 8-byte
} ;

int main(void) {
  struct aaa var1;
  node var2;


  var1.data = 111;
  var1.next = NULL;

  printf("var1.next = %p \n", var1.next);
  printf("var1.data = %d \n", var1.data);

  printf("sizeof(var1) = %ld \n", sizeof(var1));

  var2.data = 111;
  var2.next = NULL;

  printf("var2.next = %p \n", var2.next);
  printf("var2.data = %d \n", var2.data);

  printf("sizeof(var2) = %ld \n", sizeof(var2));

}
```

```c
typedef struct node node;

struct node {
  int   data;   // 4-byte
  node *next;   // 8-byte
} ;

int main(void) {
  node n1, n2, n3;
  node *sp = NULL;
  node *ep = NULL;
  node *p  = NULL;

  sp = &n1;
  ep = &n3;

  n1.data = 111;
  n1.next = &n2;

  n2.data = 222;
  n2.next = &n3;

  n3.data = 333;
  n3.next = NULL;

  printf("&n1 = %p  sizeof(n1)=%ld \n", &n1, sizeof(n1));
  printf("&n2 = %p  sizeof(n2)=%ld \n", &n2, sizeof(n2));
  printf("&n3 = %p  sizeof(n3)=%ld \n", &n3, sizeof(n3));

  printf("sp  = %p  \n", sp);
  printf("ep  = %p  \n", ep);
```


sp= &n1;
sp
&n1
data
next
&n1
data
111
next
&n2
&n2
data
next
&n2
data
222
next
&n3
ep= &n3
ep
&n3
data
next
NULL
&n3
data
333
next
NULL

```c
  printf("n1.data = %d \n", n1.data);
  printf("n1.next = %p \n", n1.next);

  printf("n2.data = %d \n", n2.data);
  printf("n2.next = %p \n", n2.next);

  printf("n3.data = %d \n", n3.data);
  printf("n3.next = %p \n", n3.next);

  printf("-------------------------\n");

  p = sp;
  printf("p = sp \n");
  printf("\t\tp->data = %d \n", p->data);
  printf("\t\tp->next = %p \n", p->next);

  p = p->next;
  printf("p = p->next \n");
  printf("\t\tp->data = %d \n", p->data);
  printf("\t\tp->next = %p \n", p->next);

  p = p->next;
  printf("p = p->next \n");
  printf("\t\tp->data = %d \n", p->data);
  printf("\t\tp->next = %p \n", p->next);

  p = p->next;
  printf("p = p->next \n");


  printf("-------------------------\n");

  p = sp;

  while (p) {
    printf("\t\tp->data = %d \n", p->data);
    printf("\t\tp->next = %p \n", p->next);

    p = p->next;
  }


}
```


sp = &n1;
sp
&n1
data
111
next
&n2
&n2
data
222
next
&n3
&n3
data
333
next
NULL
p
ep
ep = &n3
① p = sp;
② p = p->next;
③ p = p->next;
④ p = p->next;
①
②
③
④

```
&n1 = 0x7ffd7c981fc0  sizeof(n1)=16
&n2 = 0x7ffd7c981fd0  sizeof(n2)=16
&n3 = 0x7ffd7c981fe0  sizeof(n3)=16
sp  = 0x7ffd7c981fc0
ep  = 0x7ffd7c981fe0
n1.data = 111
n1.next = 0x7ffd7c981fd0
n2.data = 222
n2.next = 0x7ffd7c981fe0
n3.data = 333
n3.next = (nil)
-------------------------
p = sp
                p->data = 111
                p->next = 0x7ffd7c981fd0
p = p->next
                p->data = 222
                p->next = 0x7ffd7c981fe0
p = p->next
                p->data = 333
                p->next = (nil)
p = p->next
-------------------------
                p->data = 111
                p->next = 0x7ffd7c981fd0
                p->data = 222
                p->next = 0x7ffd7c981fe0
                p->data = 333
                p->next = (nil)
```

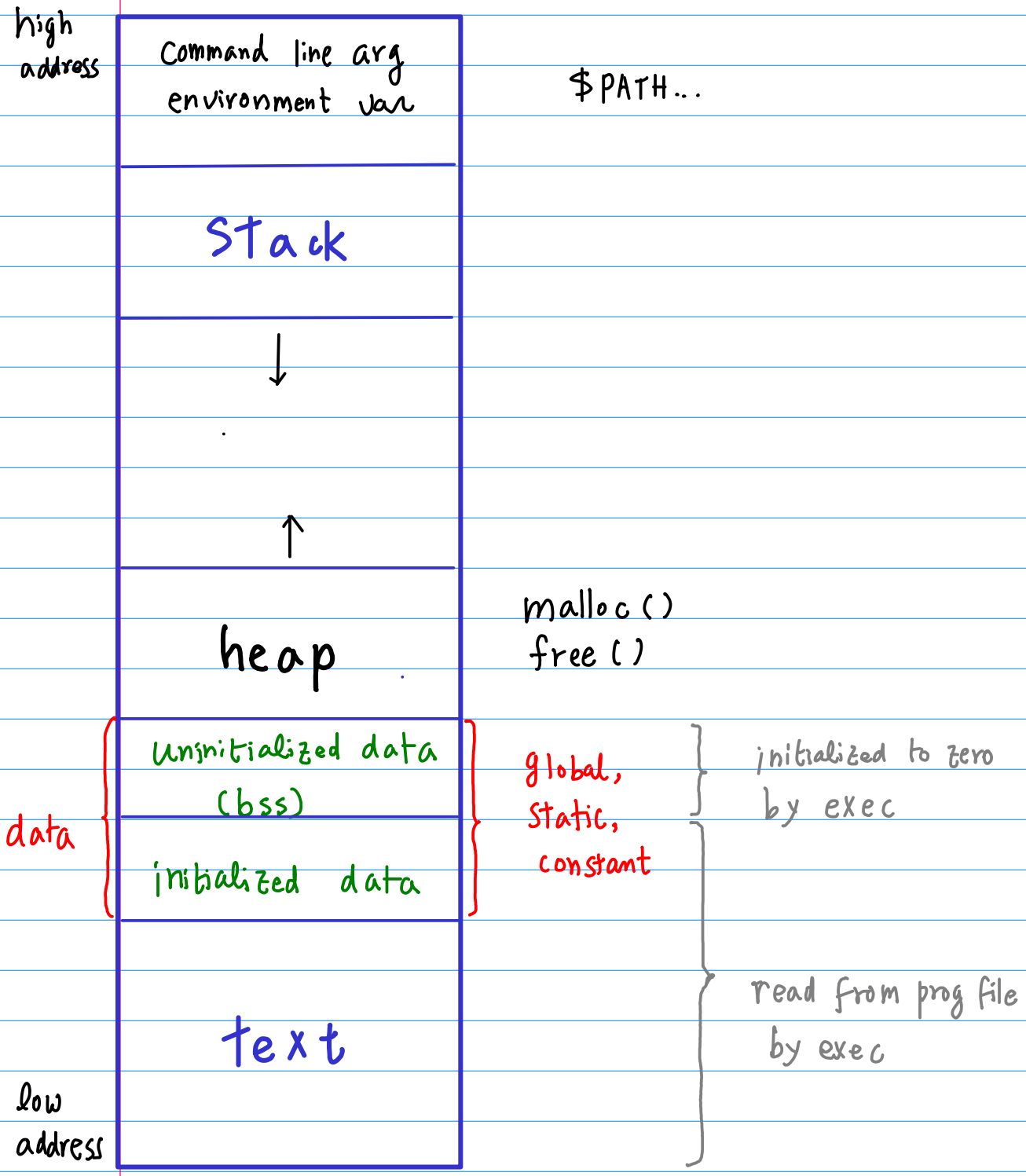
Pointer Returning Function

```c
#include <stdio.h>

int *func() {
  static int cnt=0;

  cnt++;

  return &cnt;
}


int main(void) {
  int *p;

  p = func();
  p = func();
  p = func();

  printf("*p= %d \n", *p);

  *p = 0;

  p = func();

  printf("*p= %d \n", *p);

}
```

Without the **static** keyword, cnt becomes a automatic local variable.

Local variables vanishes after returning to the caller

the corresponding stack frame are popped off

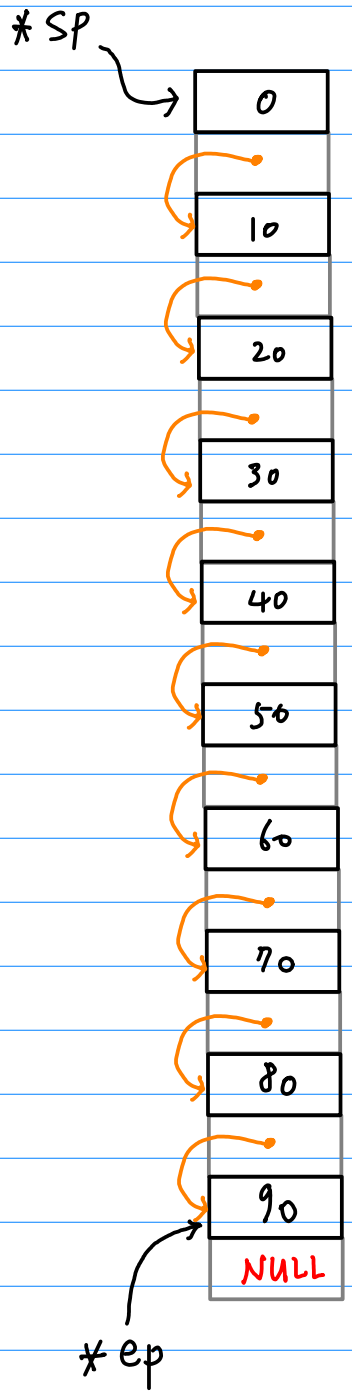


When a program is loaded into memory, it's organized into different segments. One of the segment is

Initialized data segment: All the global, static and constant data are stored here.
Uninitialized data segment(BSS): All the uninitialized data are stored in this segment.

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

//.............................
typedef struct node node;

struct node {
  int data;
  node *next;
};
//.............................


void init(node **sp, node **ep) {
  int   i= 0;
  node *p;

  do {
    p = malloc( sizeof(node) );

    p->data = 10 * i++;
    p->next = NULL;

    if (*sp == NULL) *sp = p;
    if (*ep == NULL) *ep = p;
    else {
      (*ep)->next = p;
      (*ep) = p;
    }
  } while (i<10);
}
```

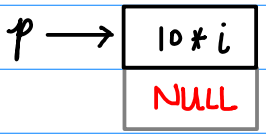
```c
int main(void) {
  node *p = NULL;   // current pointer
  node *sp= NULL;   // start pointer
  node *ep= NULL;   // end pointer

  init( &sp, &ep );

  p = sp;
  while (p) {
    printf("p->data = %d\n", p->data);

    p = p->next;
  }

}
```



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

//.............................
typedef struct node node;

struct node {
  int data;
  node *next;
};
//.............................


void init(node **sp, node **ep) {
  int   i= 0;
  node *p;

  do {
    p = malloc( sizeof(node) );

    p->data = 10 * i++;
    p->next = NULL;

    if (*sp == NULL) *sp = p;
    if (*ep == NULL) *ep = p;
    else {
      (*ep)->next = p;
      (*ep) = p;
    }
  } while (i<10);
}
```

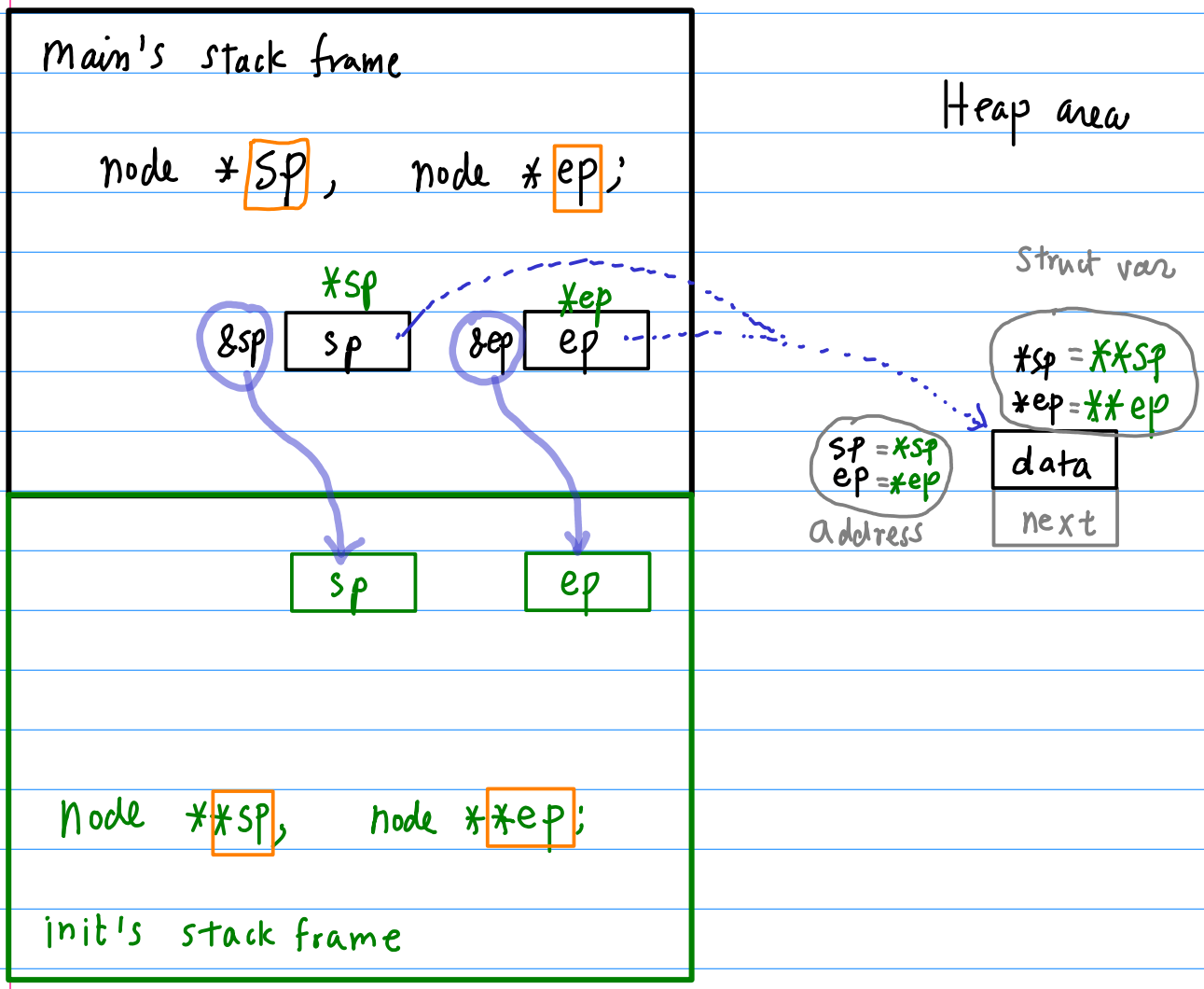
node type structure

in the heap area
preserve a location



Passing By Reference

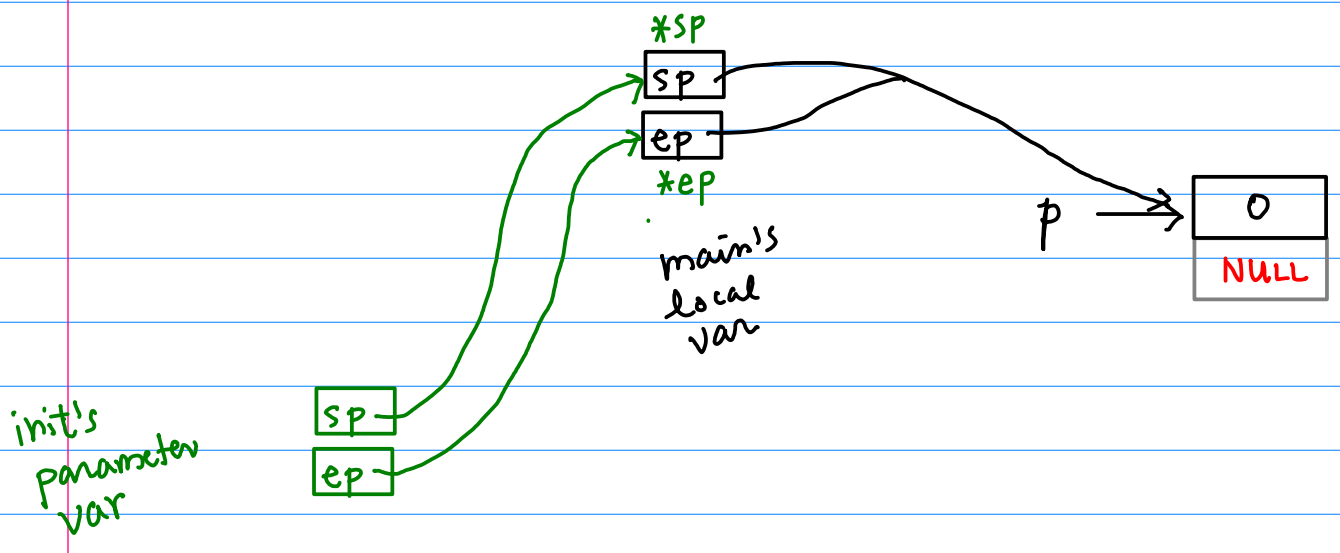
```
init( &sp, &ep );
```

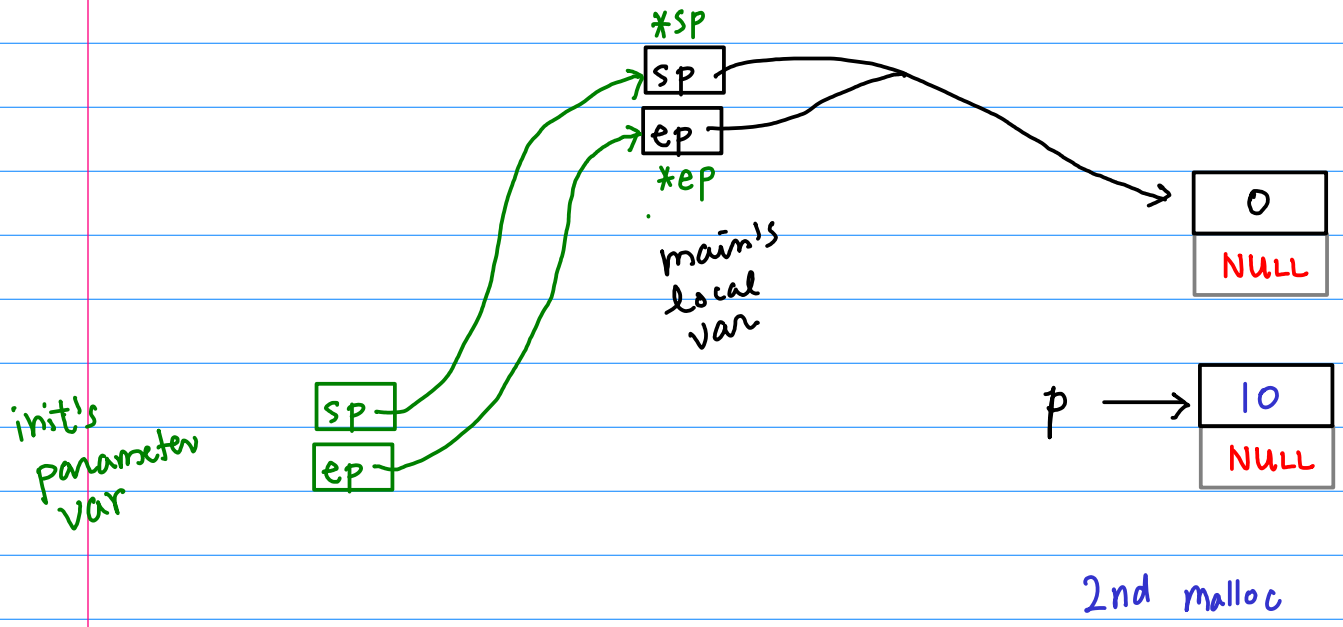



```
void init(node **sp, node **ep)
```

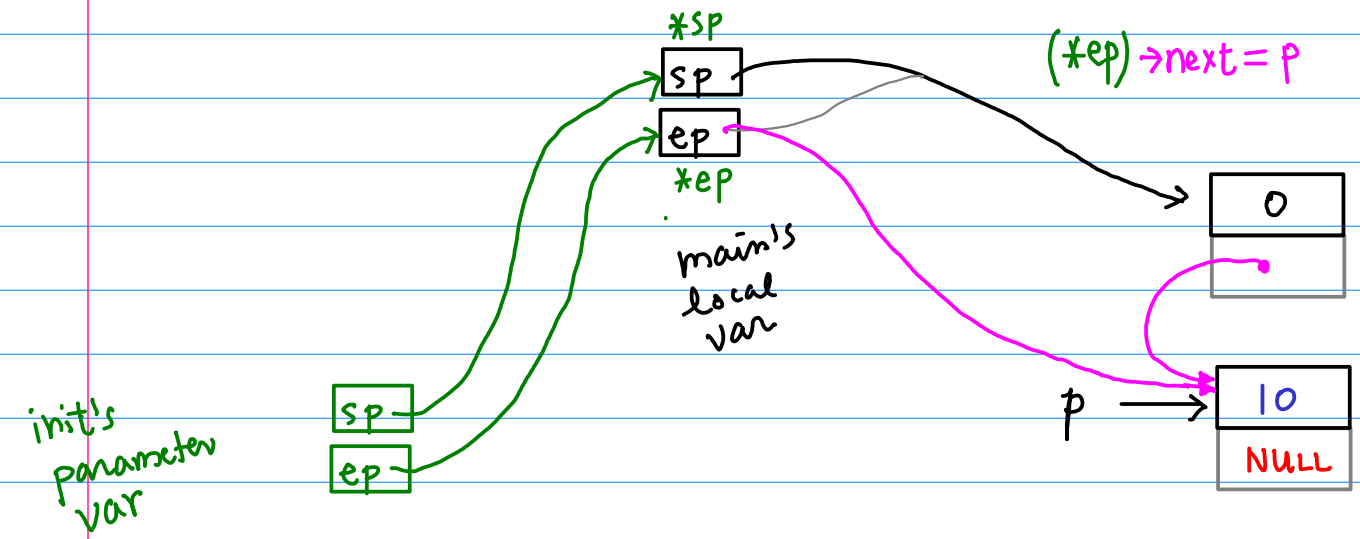
```c
#include <stdio.h>
#include <stdlib.h>

//.............................
typedef struct node node;

struct node {
  int data;
  node *next;
};
//.............................


void init(node **sp, node **ep) {
  int   i= 0;
  node *p;

  do {
    p = malloc( sizeof(node) );

    p->data = 10 * i++;
    p->next = NULL;

    if (*sp == NULL) *sp = p;
    if (*ep == NULL) *ep = p;
    else {
      (*ep)->next = p;
      (*ep) = p;
    }
  } while (i<10);
}
```

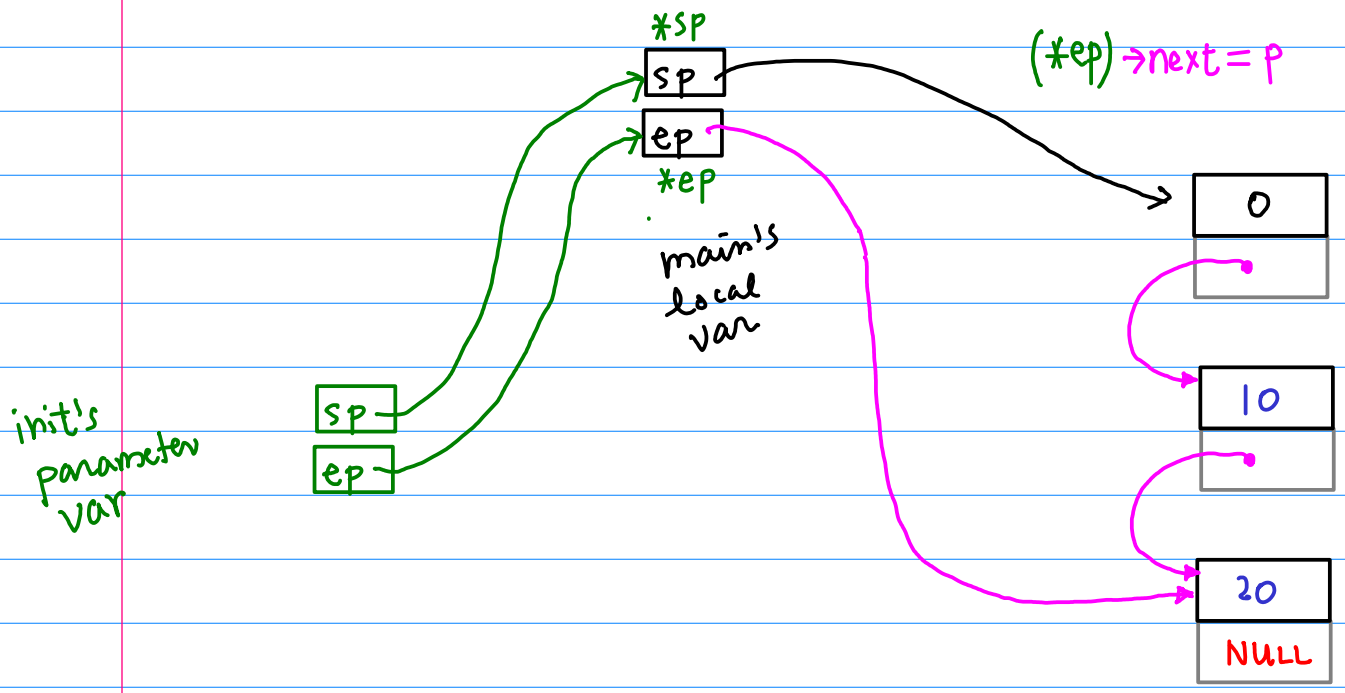
only once at the first time

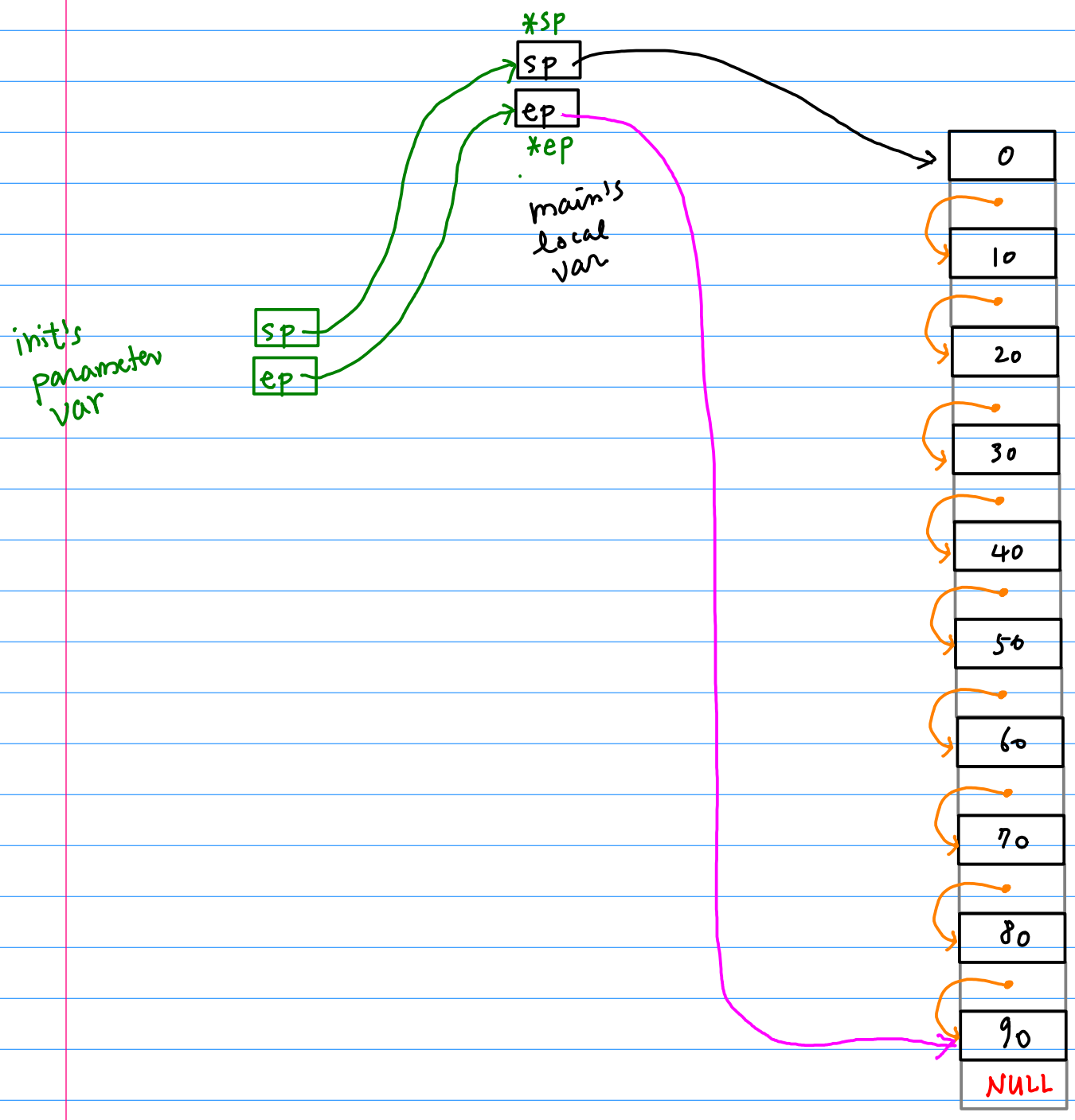


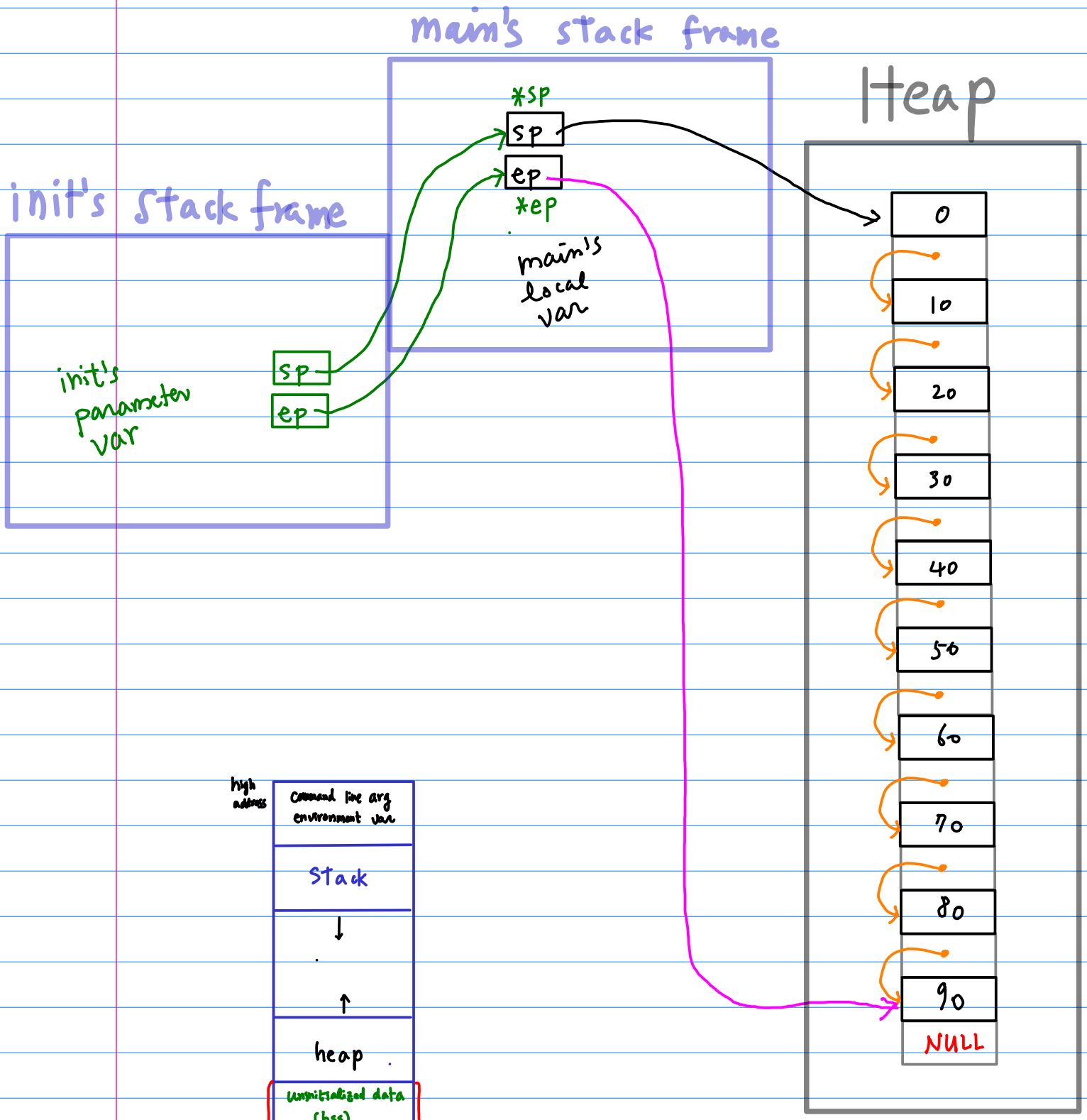



```c
do {
  p = malloc( sizeof(node) );

  p->data = 10 * i++;
  p->next = NULL;

  if (*sp == NULL) *sp = p;
  if (*ep == NULL) *ep = p;
  else {
    (*ep)->next = p;
    (*ep) = p;
  }
} while (i<10);
```

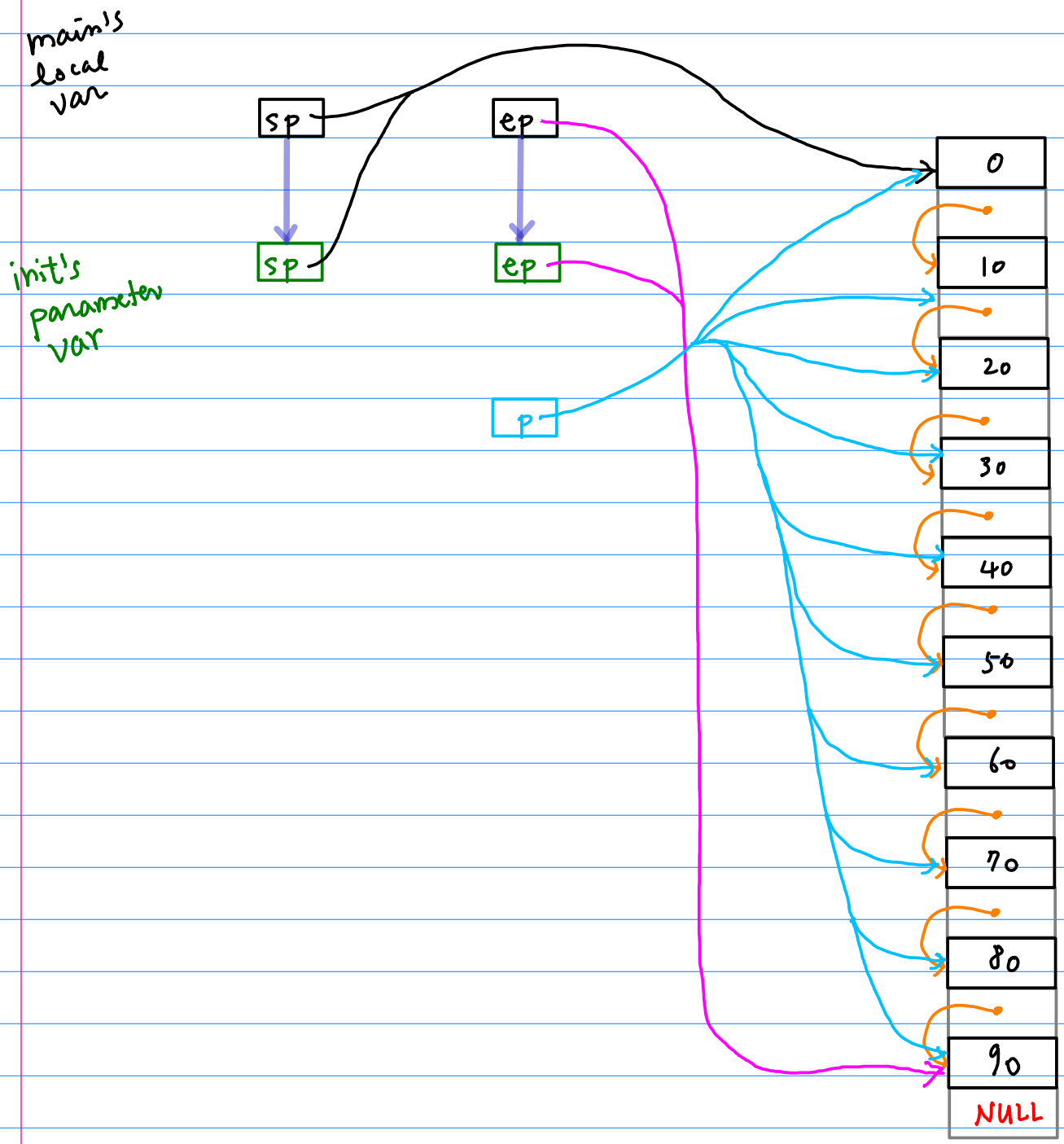



*sp
sp
ep
*ep
(*ep)->next = p
main's local var
0
10
20
NULL
init's parameter var
sp
ep


*sp
sp
ep
*ep
main's local var
init's parameter var
sp
ep
0
10
20
30
40
50
60
70
80
90
NULL


main's stack frame
*sp
sp
ep
*ep
main's local var
Heap
0
10
20
30
40
50
60
70
80
90
NULL
init's stack frame
init's parameter var
sp
ep
high address
Command line arg environment var
Stack
heap
uninitialized data (bss)
initialized data
data
text
low address

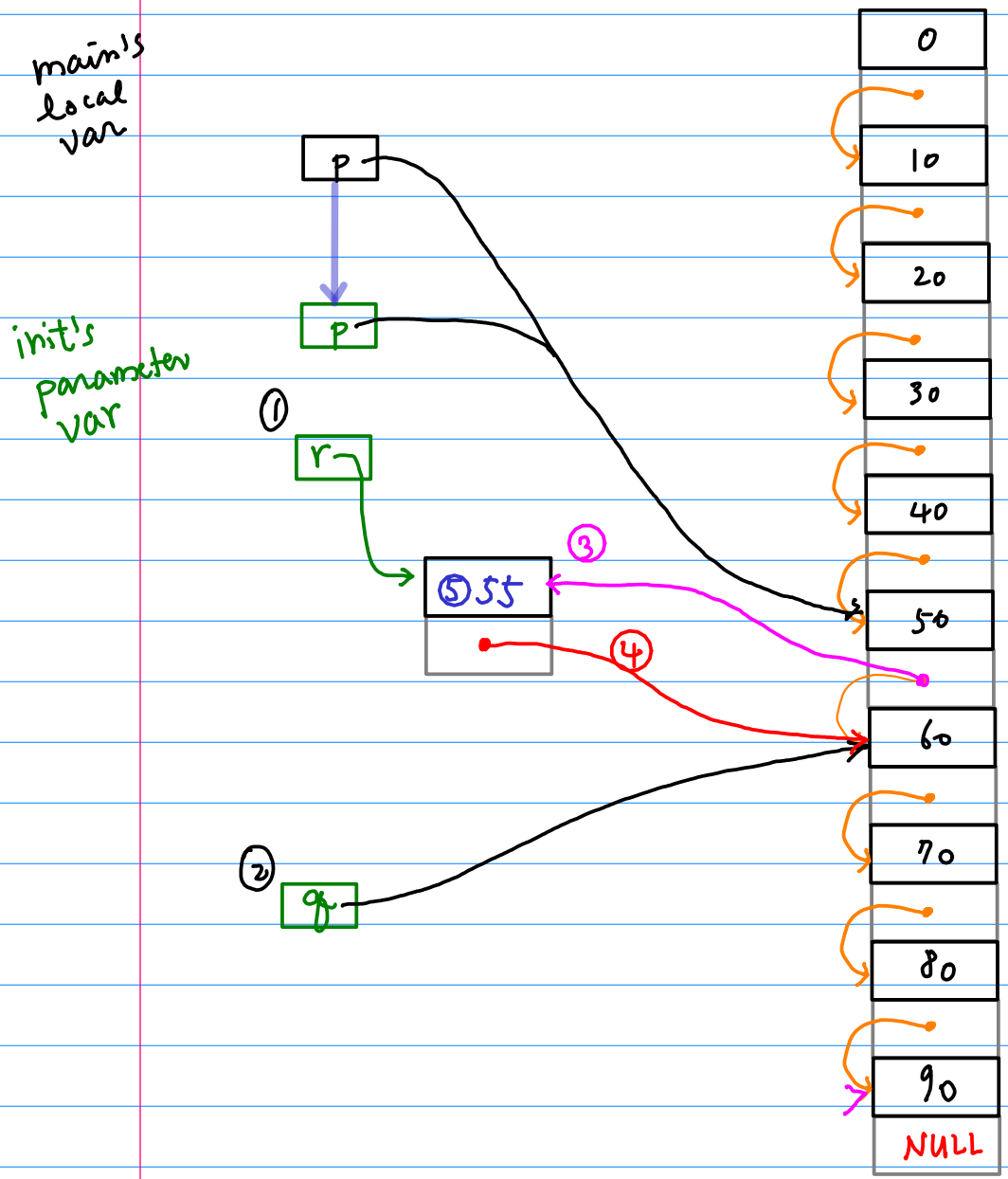
```c
node *find(node *sp, int key) {
  node *p = sp;

  while (p) {
    if (p->data == key) return p;
    p = p->next;
  }

  return NULL;
}
```



insert a node after (p)

```c
//-----------------------------------
void insert(node *p, int key) {
  node *q, *r;

① r = malloc( sizeof(node) );

② q = p->next;

③ p->next = r;
④ r->next = q;
⑤ r->data = key;

}
```

```c
int main(void) {
  node *p = NULL;   // current pointer
  node *sp= NULL;   // start pointer
  node *ep= NULL;   // end pointer

  init( &sp, &ep );

  p = find( sp, 50);

  printf("p->data = %d \n", p->data);

  insert( p, 55 );
  display( sp );

  delete( p );
  display( sp );

}
```



delete the node after p

```c
//-----------------------------------
void delete(node *p) {
  node *q, *r;

① q = p->next;
② r = q->next;

③ p->next = r;

  free( q );
}
```

```c
int main(void) {
  node *p = NULL;   // current pointer
  node *sp= NULL;   // start pointer
  node *ep= NULL;   // end pointer

  init( &sp, &ep );

  p = find( sp, 50);

  printf("p->data = %d \n", p->data);

  insert( p, 55 );
  display( sp );

  delete( p );
  display( sp );

}
```