Structs

Struct stores fields of different types contiguously in memory

Structure

- Array: a block of n consecutive elements of the same type.

- Struct: a collection of elements of diffferent types.

Structure

```
struct student {
    int id;
    char *name;
};
```

Fields of a struct are allocated next to each other, but there may be gaps (padding) between them.

Structure

```
struct student {
    int id;
    char *name;
};

struct student t;
```

define variable t with type “struct student”

Structure

```
struct student {
    int id;
    char *name;
};

struct student t;

t.id = 1024;
t.name = “alice”;
```

Access the fields of this struct

Typedef

```
typedef struct {
    int id;
    char *name;
} student;
```

~~struct~~ student t;

Pointer to struct

```
typedef struct {
    int id;
    char *name;
} student;

student t = {1023, “alice”};
student *p = &t;

p->id = 1023;
p->name = “bob”;
printf(“%d %s\n”, t.id, t.name\n”);
```

Mallocs

Allocates a chunk of memory dynamically

Recall memory allocation for global and local variables

- Global variables are allocated space before program execution.

- Local variables are allocated when entering a function and de-allocated upon its exit.

Malloc

Allocate space dynamically and flexibly:

– malloc: allocate storage of a given size

– free: de-allocate previously malloc-ed storage

```
void *malloc(size_t size);
```

A void pointer is a pointer that has no associated data type with it. A void pointer can hold address of any type and can be casted to any type.

```
void free(void *ptr);
```

Malloc

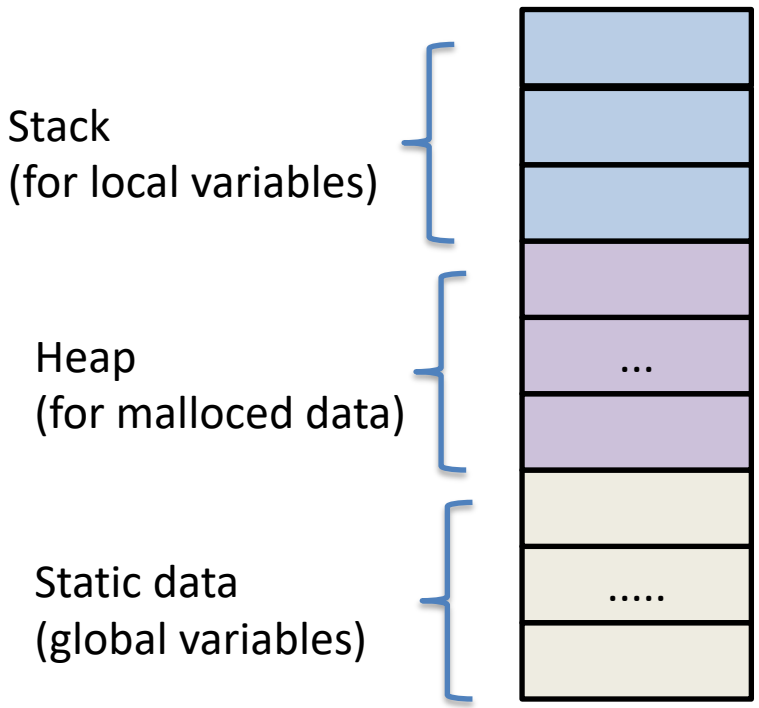
Malloc is implemented as a C library

```
#include <stdlib.h>

int *newArray(int n) {
   int *p;
   p = (int*)malloc(sizeof(int) * n);
   return p;
}
```

Conceptual view of a C program's memory at runtime

- Separate memory regions for global, local, and malloc-ed.



We will refine this simple view in later lectures

Linked list in C: insertion

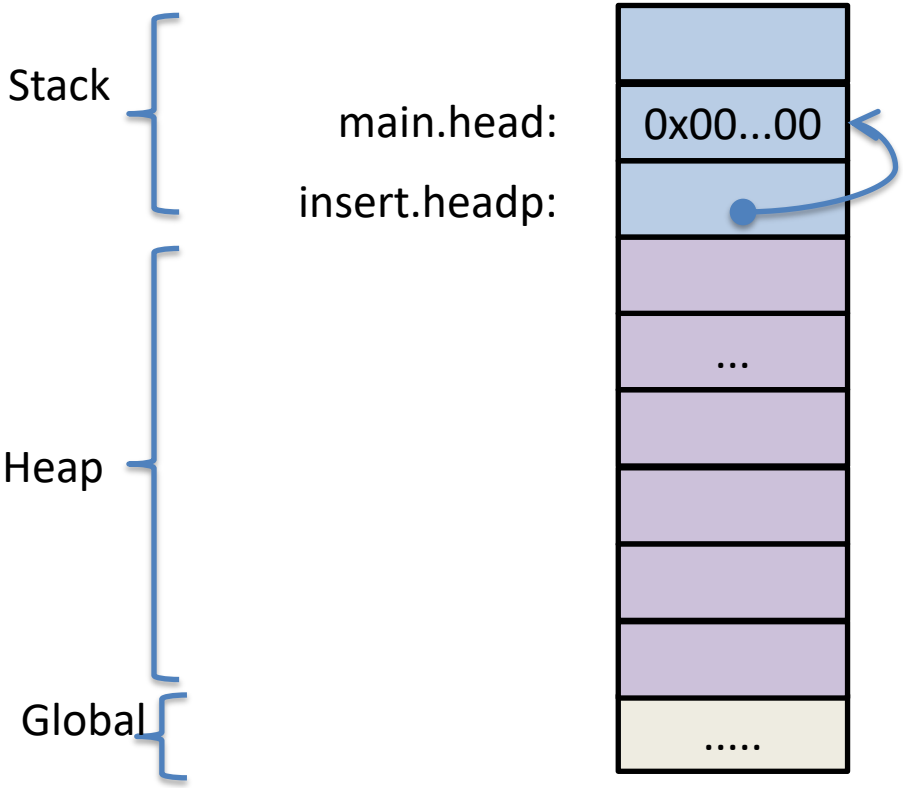
```
typedef struct {
     long val;
     struct node *next;
 }node;

// insert val into linked list to the head of the linked
// list and return the new head of the list in *head
void
insert(node **head, long val) {

}

int main() {
    node *head = NULL;
    for (long i = 10; i < 13; i++)
       insert(&head, i);
}
```

Inserting into a linked list

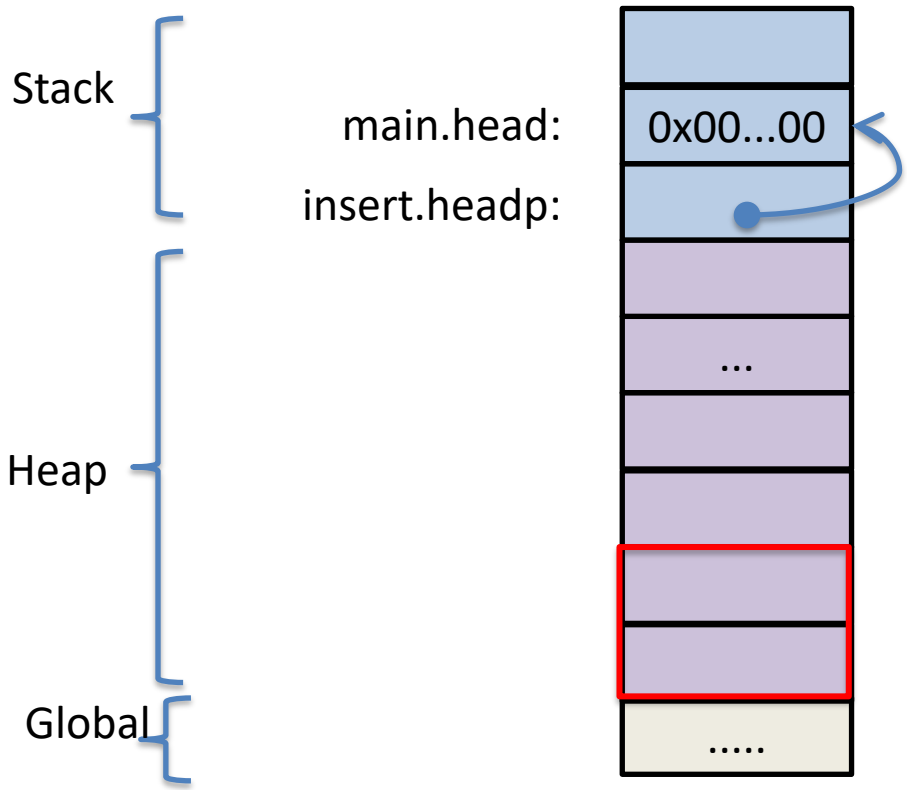



```
void insert(node **headp, long val)
{

}

int main()
{
    node *head = NULL;
    for (long i = 10; i < 13; i++) {
        insert(&head, i);
    }
}
```

1st insert call



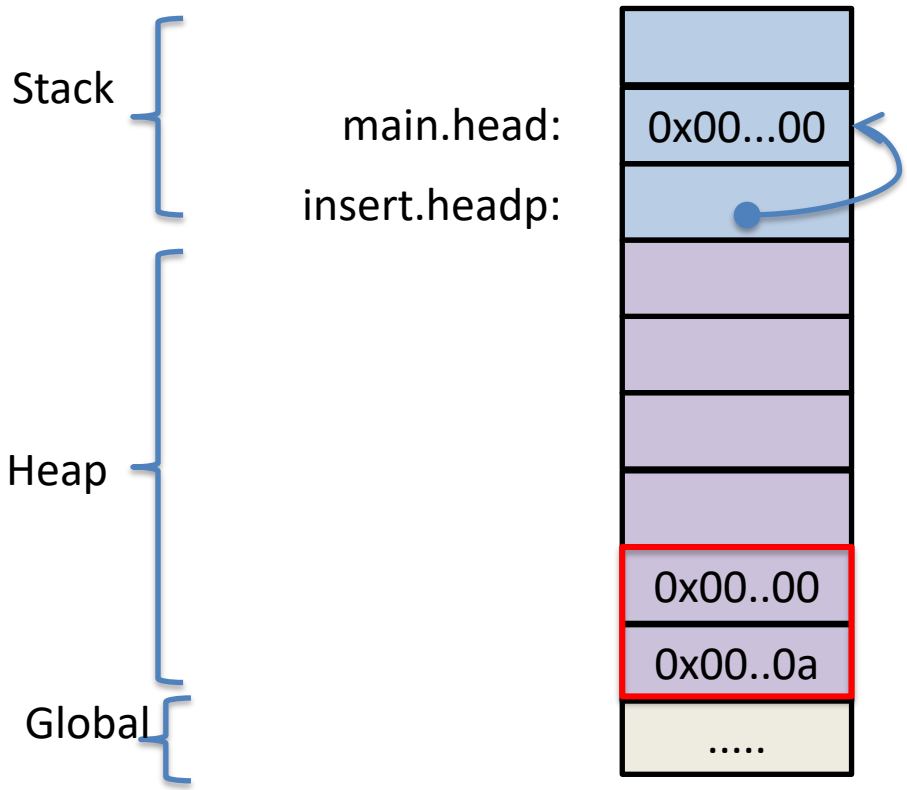

```
void insert(node **headp, long val)
{
    node *n;
    n = (node *)malloc(sizeof(node));



}

int main()
{
    node *head = NULL;
    for (long i = 10; i < 13; i++) {
        insert(&head, i);
    }
}
```

1st insert call

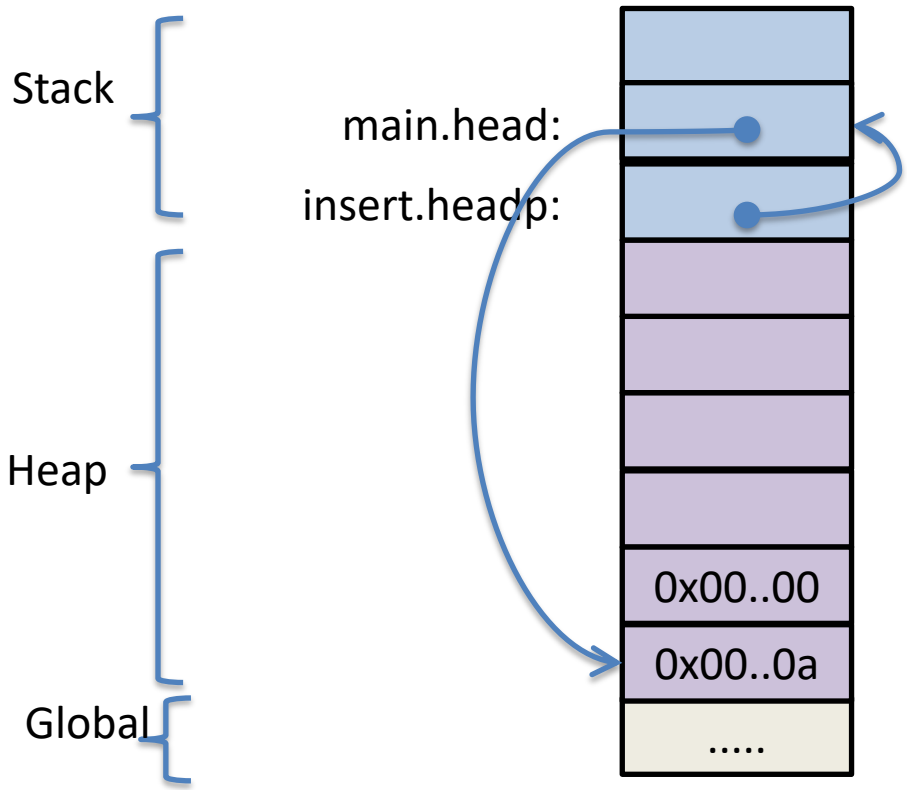



```
void insert(node **headp, long val)
{
    node *n;
    n = (node *)malloc(sizeof(node));
    n->val = val;
    n->next = *headp;

}

int main()
{
    node *head = NULL;
    for (long i = 10; i < 13; i++) {
        insert(&head, i);
    }
}
```

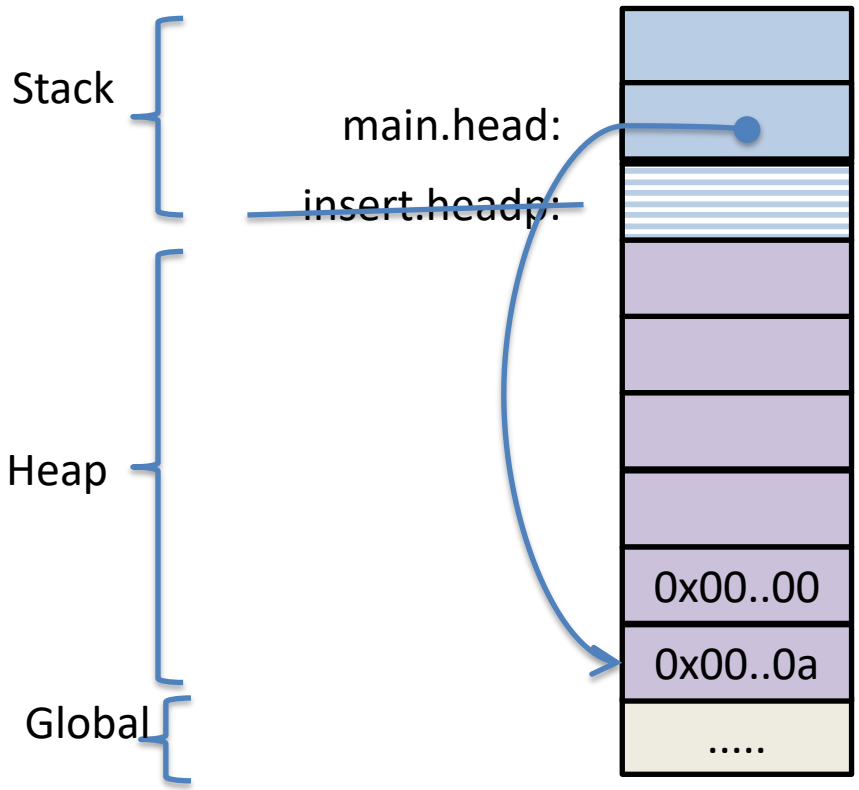
1st insert call




```
void insert(node **headp, long val)
{
    node *n;
    n = (node *)malloc(sizeof(node));
    n->val = val;
    n->next = *headp;
    *headp = n;
}

int main()
{
    node *head = NULL;
    for (long i = 10; i < 13; i++) {
        insert(&head, i);
    }
}
```

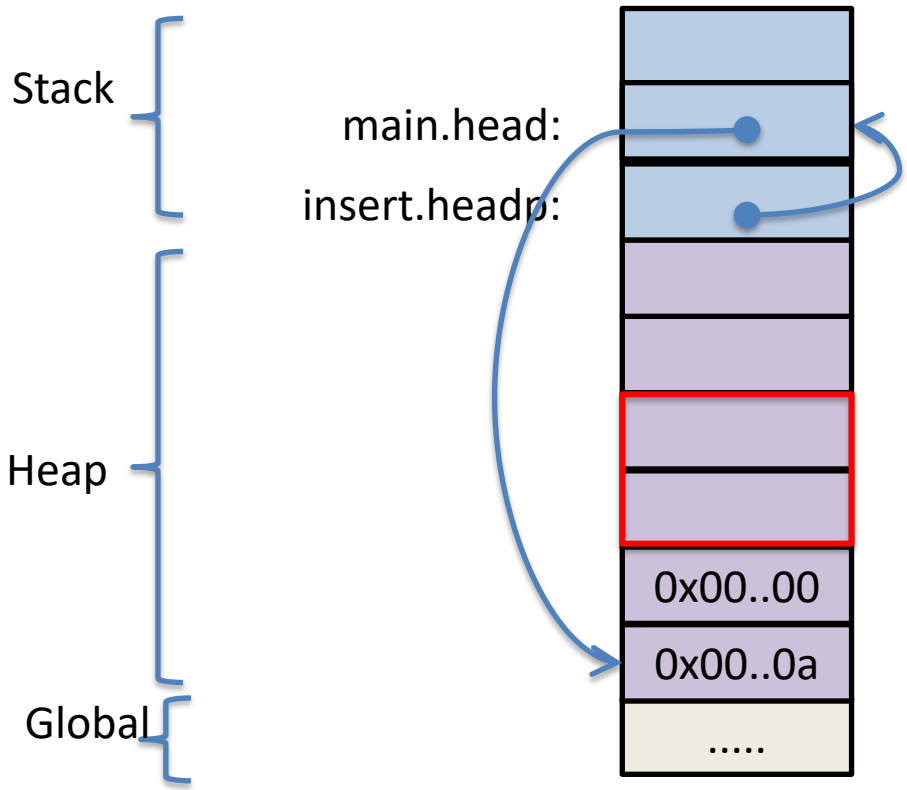
after 1st insert call




```
void insert(node **headp, long val)
{
    node *n;
    n = (node *)malloc(sizeof(node));
    n->val = val;
    n->next = *headp;
    *headp = n;
}

int main()
{
    node *head = NULL;
    for (long i = 10; i < 13; i++) {
        insert(&head, i);
    }
}
```

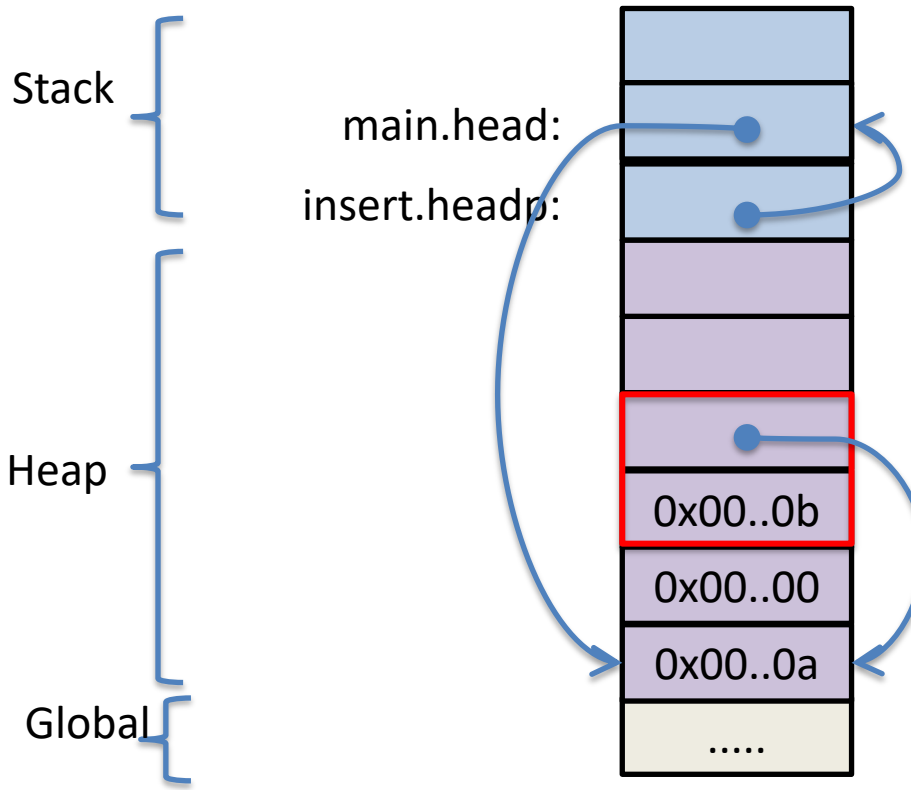
2nd insert call




```
void insert(node **headp, long val)
{
    node *n;
    n = (node *)malloc(sizeof(node));
    n->val = val;
    n->next = *headp;
    *headp = n;
}

int main()
{
    node *head = NULL;
    for (long i = 10; i < 13; i++) {
        insert(&head, i);
    }
}
```

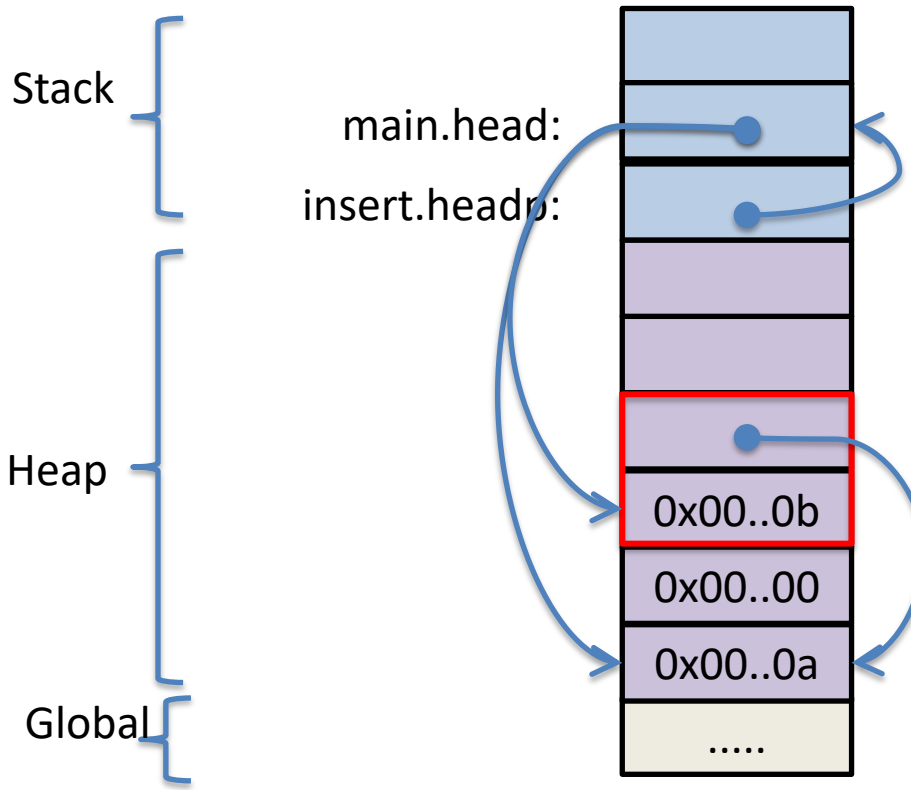
2nd insert call




```
void insert(node **headp, long val)
{
    node *n;
    n = (node *)malloc(sizeof(node));
    n->val = val;
    n->next = *headp;
    *headp = n;
}

int main()
{
    node *head = NULL;
    for (long i = 10; i < 13; i++) {
        insert(&head, i);
    }
}
```

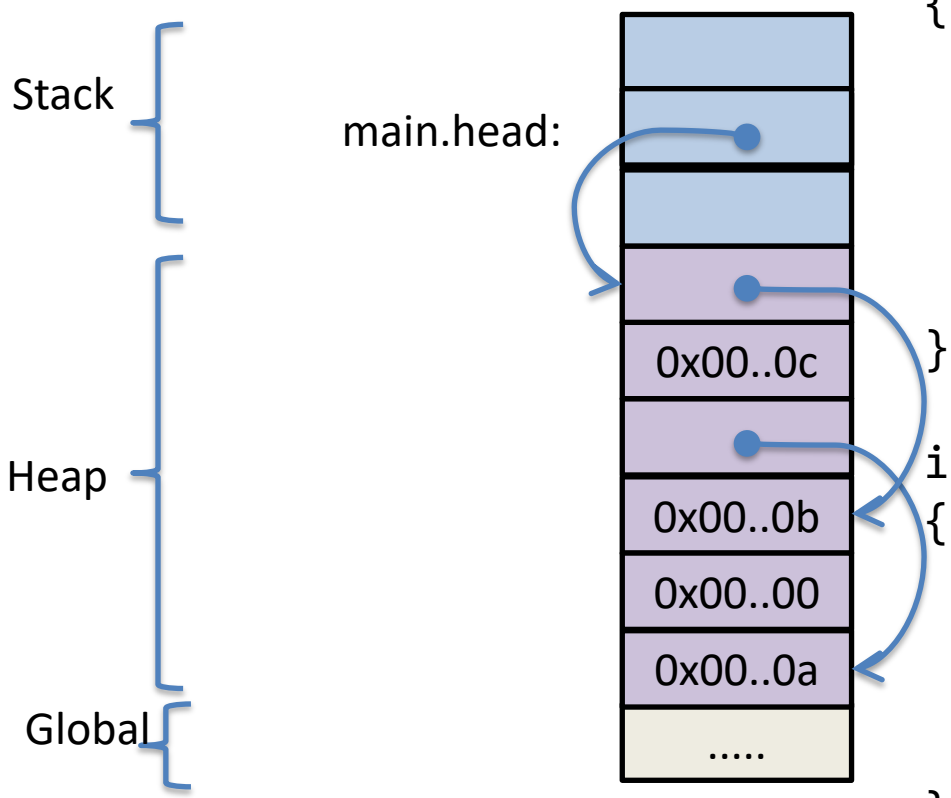
2nd insert call




```
void insert(node **headp, long val)
{
    node *n;
    n = (node *)malloc(sizeof(node));
    n->val = val;
    n->next = *headp;
    *headp = n;
}

int main()
{
    node *head = NULL;
    for (long i = 10; i < 13; i++) {
        insert(&head, i);
    }
}
```

after 3rd call




```
void insert(node **headp, long val)
{
    node *n;
    n = (node *)malloc(sizeof(node));
    n->val = val;
    n->next = *headp;
    *headp = n;
}

int main()
{
    node *head = NULL;
    for (long i = 10; i < 13; i++) {
        insert(&head, i);
    }
}
```