# Struct, malloc

Jinyang Li

# Structs

Struct stores fields of different types contiguously in memory

# What's a struct?

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

- Struct: a collection of data of different types.
  - C has no support for object oriented programming
  - You can view structs as rudimentary "objects" without associated member functions

# Define and access struct

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

struct student t;

t.id = 1024;
t.name = “alice”;
```

struct declaration:
a struct’s fields are contiguous in memory, with potential gaps, aka padding, in between.

define variable t with type “struct student”

Access fields of the struct variable

# typedef struct

```
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 = 1024;
p->name = “bob”;
printf(“%d %s\n”, t.id, t.name);
```

P->name is shorthand for (*p).name

# void pointer

```
void memset_zero(void *p, int n)
{
    char *q = (char *)p;
    for (int i = 0; i < n; i++)
        q[i] = 0;
}

int main()
{

    student s;
    memset_zero(&s, sizeof(s));

    teacher t;
    memset_zero(&t, sizeof(t));
}
```

How to make memset_zero work with different variable types?

memset is part of stdlib, type: "man memset"

# Malloc

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.

# Allocating a new array?

```
int *newArray(int n) {
   int arr[n];
   return arr;
}

int main() {
   int *r;
   r = newArray(1000);
   //do something with the array
   …
}
```

# What's malloc?

- A collection of stdlib functions for dynamic memory allocation:
  - malloc: allocate storage of a given size

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

  - free: de-allocate previously malloc-ed storage

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

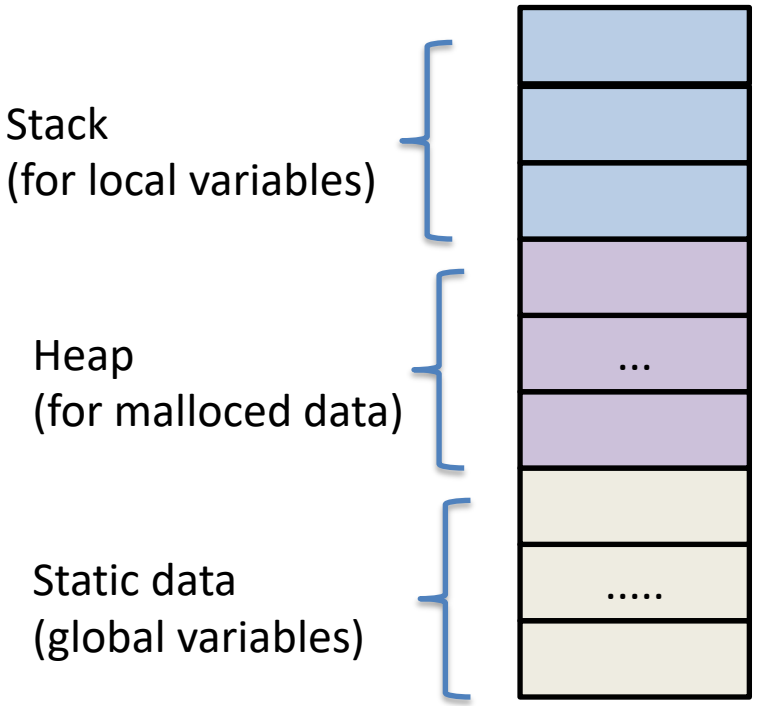
# Malloc

Malloc is implemented by C standard 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

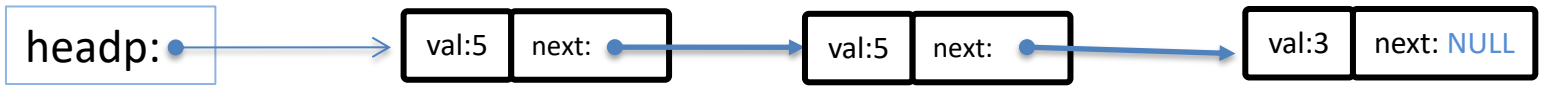
- Separate memory regions for global, local, and malloc-ed.




We will refine this simple view in later lectures

# Linked list

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




```
node *find(node *headp, int val) {
   node *n = headp;
   while (n) {
      if (n->val == val)
         break;
      n = n->next;
   }
   return n;
}
```

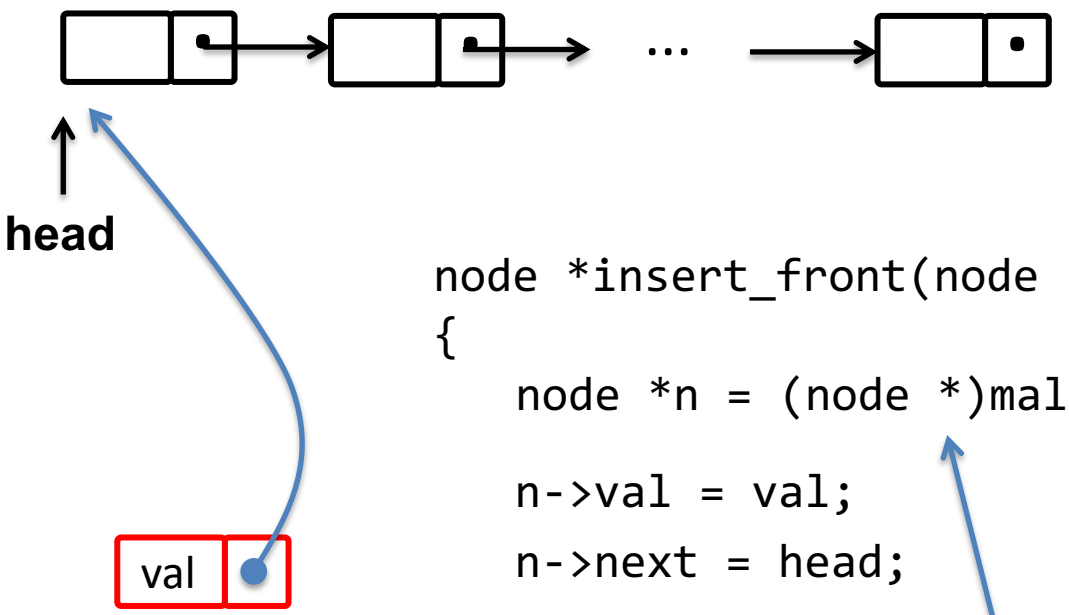
# Linked list in C: insertion

```
// insert val in the front of the linked list
// returns new head
node *insert_front(node *headp, int val) {



}



int main() {
    node *headp = NULL;
    for (int i = 0; i < 3; i++)
       headp = insert_front(headp, i);
}
```

# Linked list in C: insertion

```
// insert val in the front of the linked list
// returns new head
void insert_front(node **headdp, int val)
{


}



int main() {
    node *headp = NULL;
    for (int i = 0; i < 3; i++)
        insert_front(&headp, i);
}
```

# Inserting into a linked list




```
node *insert_front(node *head, int val)
{
    node *n = (node *)malloc(sizeof(node));

    n->val = val;
    n->next = head;
    return n;

}
```

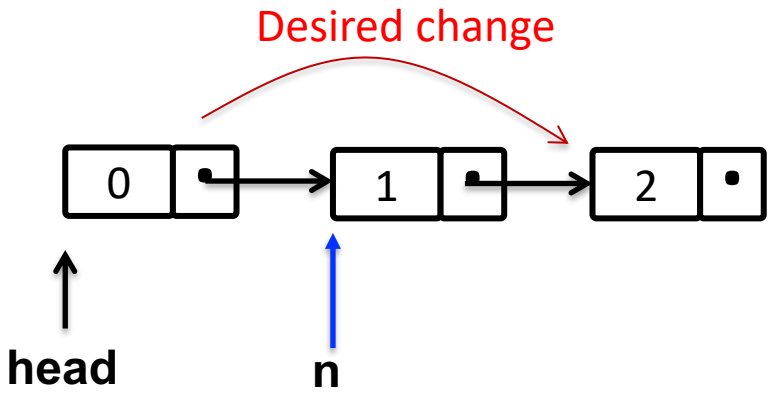
replace line with following?

```
node new_node;
node *n = &new_node;
```

# Linked list in C: removal

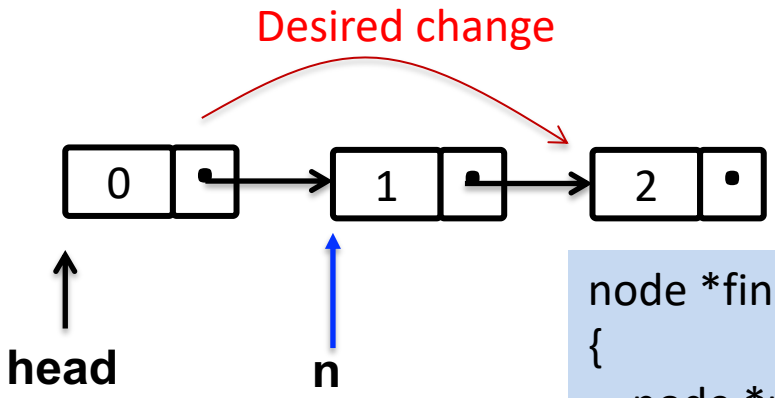
```
// remove node with val from linked list, return the new
// head of the list.
node* remove(node *head, int val)
{

}

int main() {
    node *head = NULL;
    for (int i = 0; i < 3; i++)
       head = insert(head, i);
    head = remove(head, 1);
}
```

# Removing from a linked list




```
node* remove(node *head, int val)
{
    node *n;
    n = find_node(head, val);
    // ???? How to get to n's predecessor?
}
```

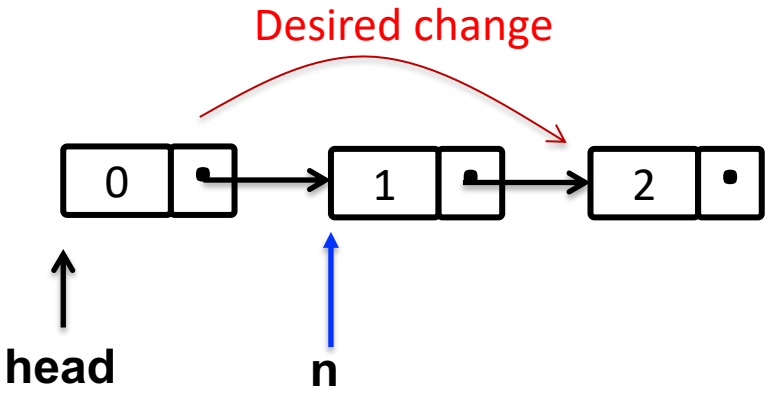
# Removing from a linked list




```
node *remove(node *head, int val) {
   node *n;
   node *pred;
   n = find(head, val, &pred);

}
```

```
node *find(node *head, int val, node **predp)
{
   node *n = head;
   node *pred = NULL;
   while (n) {
     if (n->val == val)
          break;
    pred = n;
     n = n->next;
   }
   *predp = pred;
   return n;
}
```

# Removing from a linked list




```
node *remove(node *head, int val) {
    node *n;
    node *pred;
    n = find(head, val, &pred);
    if (!n)
        return head;
    if (!pred)
        head = n->next;
    else
        pred->next = n->next;
    free(n);
    return head;
}
```

Two corner cases:

1. val is not in the list
2. n is the head

# Summary

- Struct
  - Group variables together into a primitive “object”
- Malloc
  - Allocate data on the heap
  - Must be explicitly free-ed by programmers