

# 601.220 Intermediate Programming

More linked lists

## Additional linked list operations

- `clear` - deallocates all nodes in the list, sets head pointer to null
- `add_front`
- `clear_list` (free all nodes)
- `remove_after`
- `remove_front`
- `remove_all` (remove all occurrences of a particular data value)

# Pointers are passed by value

```
// pointer_pv.c:
#include <stdio.h>

void fun1(int * ip) {
    *ip = 10;
    ip += 1; // increment the address
}

int main() {
    int a = 12;
    int * p = &a;
    printf("p points to address %p with value %d\n", (void *)p, *p);
    fun1(p); // pass p by value; changes to p will NOT affect p
    printf("p points to address %p with value %d\n", (void *)p, *p);
    return 0;
}
```

```
$ gcc -std=c99 -pedantic -Wall -Wextra pointer_pv.c
```

```
$ ./a.out
```

```
p points to address 0x7ffd92af295c with value 12
```

```
p points to address 0x7ffd92af295c with value 10
```

# Pass a pointer by reference

```
// pointer_pv.c:
#include <stdio.h>

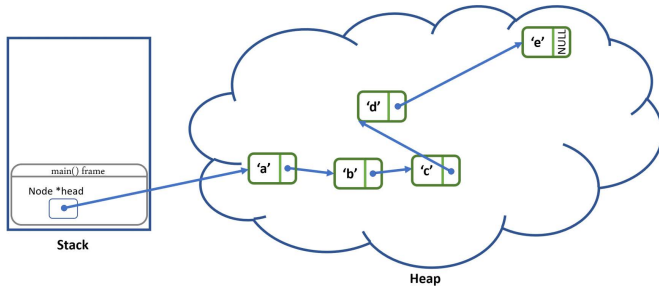
void fun1(int ** ip) {
    *ip += 1; // increment the address
}

int main() {
    int a = 12;
    int * p = &a;
    printf("p points to address %p with value %d\n", (void *)p, *p);
    fun1(&p); // pass p by value; changes to p will NOT affect p
    printf("p points to address %p with value %d\n", (void *)p, *p);
    return 0;
}

$ gcc -std=c99 -pedantic -Wall -Wextra pointer_pv.c
$ ./a.out
p points to address 0x7ffe814d3f4c with value 12
p points to address 0x7ffe814d3f50 with value -2125643952
```

# Linkedlist head

- The linked list *head* should be passed by reference if it needs to be updated



## add\_after vs. add\_front

- `void add_after(Node * node, char val);`
- `void add_front(Node ** list_ptr, char val);`
  - needs ability to modify actual head pointer (not a copy), so call with `&head` as argument

## Example add\_front call: add\_front(&head, value);

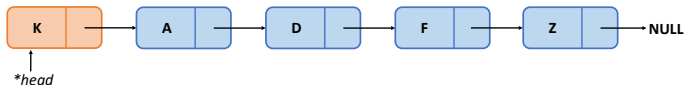
```
void add_front(Node ** list_ptr, char val) {  
    Node * n = create_node(val);  
    n->next = *list_ptr; //new node's next gets address of old first node  
    *list_ptr = n; //head pointer gets address of new node  
}
```

add\_front(&head, 'K');

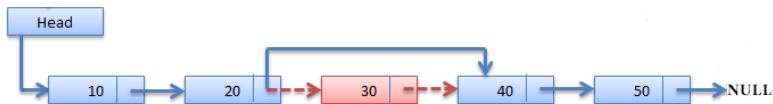
1) Create a new node



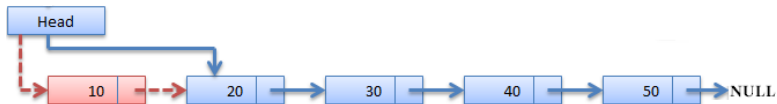
2) Add the new node in the front



# Remove Operations



```
char remove_after(Node * node);
```



```
char remove_front(Node ** list_ptr);
```



# Zoom poll!

Defintion of a Node data type:

```
typedef struct node_ {  
    char data;  
    struct node_ *next;  
} Node;
```

Consider the following function:

```
void mystery(Node **list_ptr) {  
    Node *head = *list_ptr;  
    list_ptr = list_ptr->next;  
    free(head);  
}
```

What does this function do?

- A. Correctly removes the first node for any list
- B. Correctly removes the first node of any non-empty list
- C. Has no effect
- D. The code does not compile
- E. None of the above