

Heterogeneous Data Structures

Note Title

10/15/2007

```
typedef struct p_info { /* process info */
    char name [30];
    short int id;
    int uid;
    int priority;
    struct p_info * next;
} p_info;
```

```
p_info    all [10000]; /* all processes */
p_info    * head;
p_info    * tail;
p_info    * free;
```

```

void init_p_info() {
    p_info *curr, *last;
    int i;

```

```

    head = tail = NULL;
    last = NULL;
    for (i = 0; i < 10000; i++) {
        curr = &all[i];
        curr->id = i;
        curr->next = last;
        last = curr;
    }

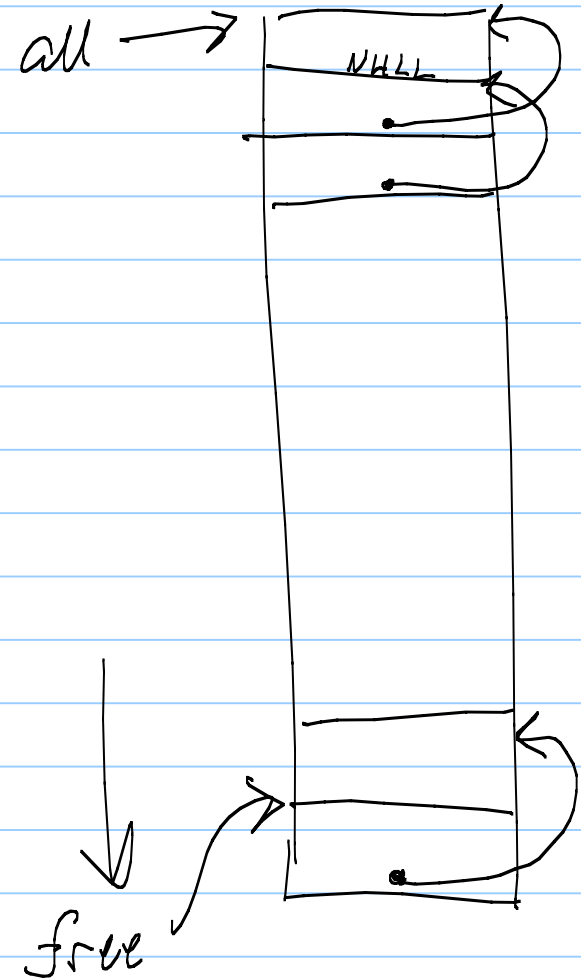
```

```

    free = curr;
    return;
}

```

(*curr).id



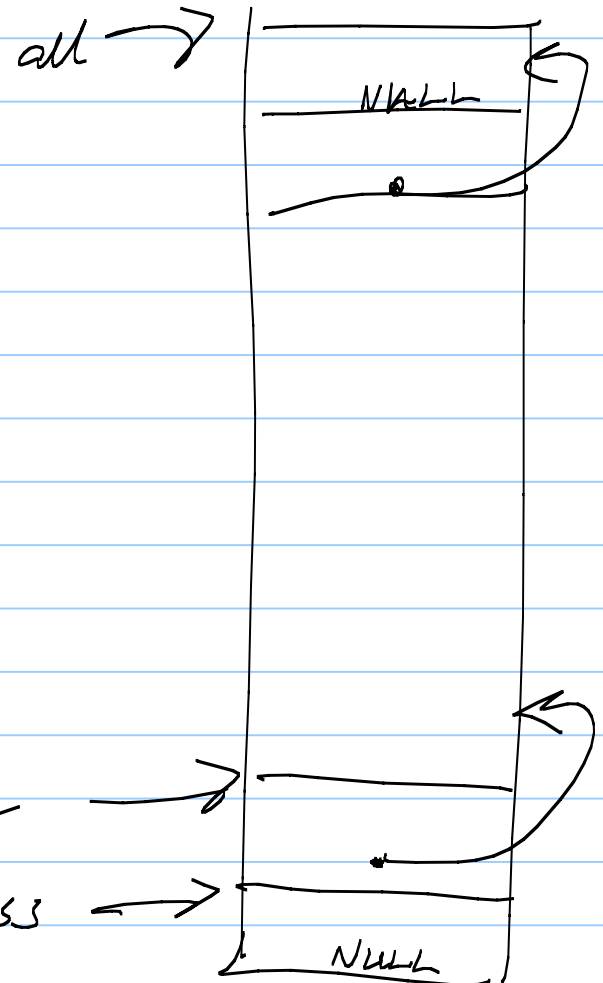
```
p_info *new_p() {  
    p_info *new;
```

```
    if (free == NULL) {  
        return (NULL);  
    }
```

```
    new = free;  
    free = free -> next;  
    new -> next = NULL;  
    return (new);  
}
```

```
p_info *process;
```

```
process = new_p();
```



```

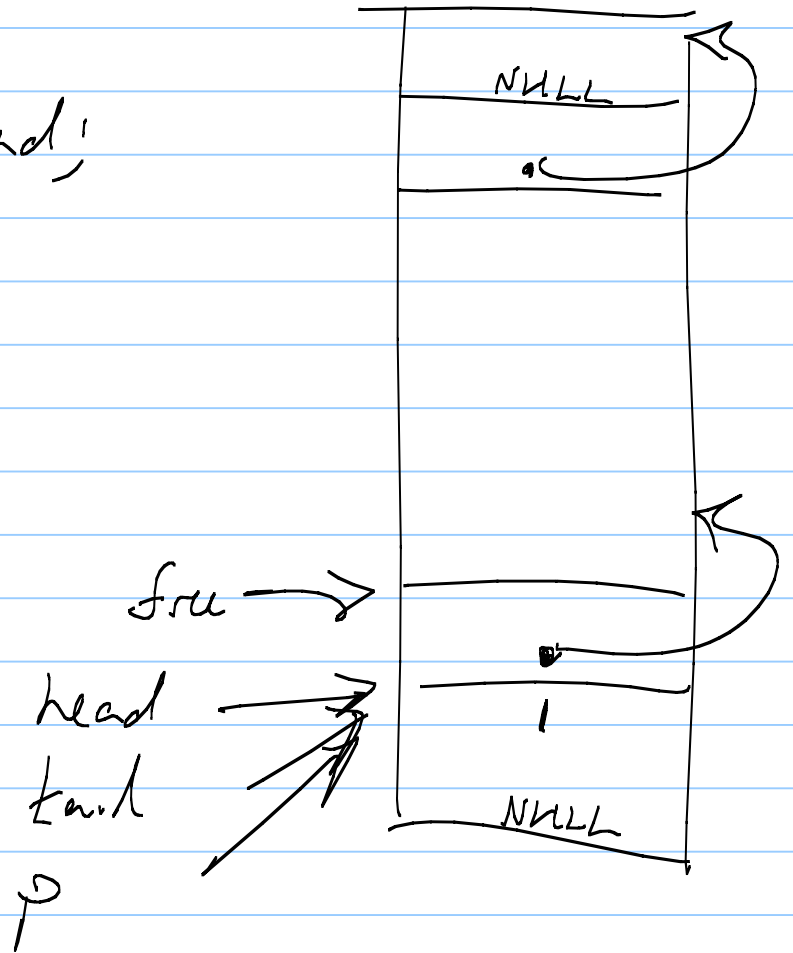
void insert_head(p_info *proc) {
    if (proc == NULL) {
        return;
    }
    proc -> next = head;
    head = proc;
    if (tail == NULL) {
        tail = proc;
    }
    return;
}

```

```

p_info *p
p = new_p();
if (p == NULL) error();
p->id = 1;
insert_head(p);

```



```
void insert_tail (p_info *proc)
```

```
if (proc == NULL) {  
    return;  
}
```

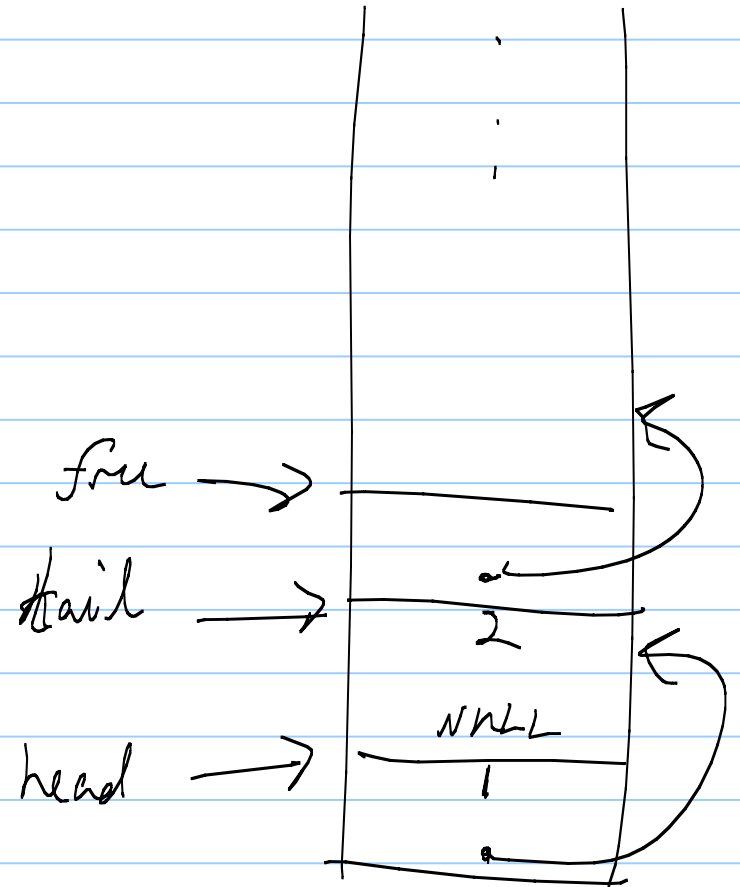
```
if (tail == NULL) {  
    head = proc;  
}
```

```
else {  
    tail->next = proc;  
}
```

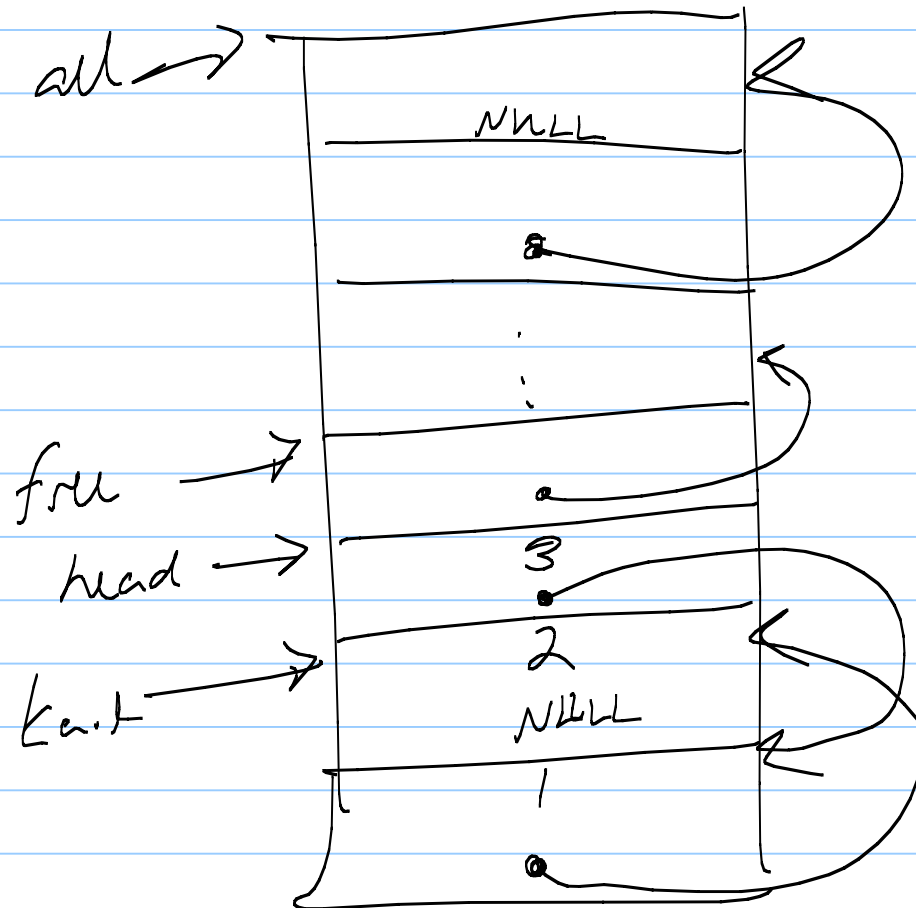
```
proc->next = NULL;  
tail = proc;  
return;  
}
```

3

```
...  
p = new p();  
p->id = 2;  
insert_tail (p);
```



insert_head() w/id 3



```
p_info * lookup (short int id) {  
    p_info * curr;  
    curr = head;  
    while (curr != NULL) {  
        if (curr->d == id)  
            break;  
        curr = curr->next;  
    }  
    return (curr);  
}
```