Lecture11: Basic Data Structure with Linked Lists
12/3/2012

Slides modified from Yin Lou, Cornell CS2022: Introduction to C

1

Administration

- Assignment #11 is on the course homepage.

2

Pointer Explained Again

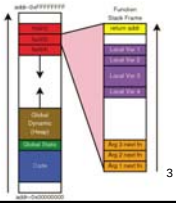
- C gives you the "memory address" where a variable's value is stored in the computer. Use "&".

```
int a = 10;;
int* b = &a;
```

- C also allows you to retrieve the value stored at a "memory address". Use "*".

```
int c = *b;
```

- What is C's memory model?
- What is the type for memory address?



3

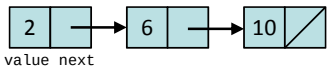
What is Data Structure?

- A way to store and organize data so that the data can be used efficiency.
- How to store records of students at NTU?
  - Record his/her name, address, student ID, courses, grades, …
  - The number of students is growing
  - May want to sort the list in a particular way

4

Linked List Implementation in C

```
struct node {
   int value;
   struct node *next;
};
```

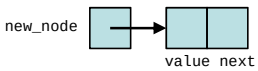



```
struct node *first = NULL;
```

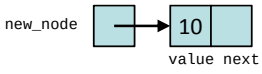
5

Create a node

```
struct node {
   int value;
   struct node *next;
};

new_node = malloc(sizeof(struct node));
```

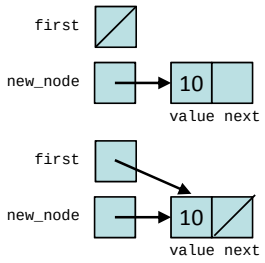



```
(*new_node).value = 10;
new_node->value = 10;
```



6

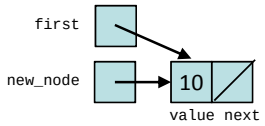
Insert a node at the Beginning of a list

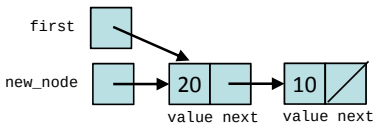



```
first = new_node;
new_node->next = NULL;
```


7

Insert another node (20) – How?

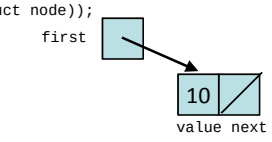


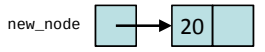


8

Insert another node (20) – How?

```
new_node = malloc(sizeof(struct node));
new_node->value = 20;
```

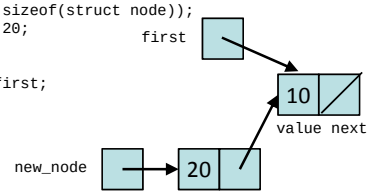




9

Insert another node (20) – How?

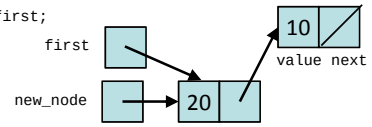
```
new_node = malloc(sizeof(struct node));
new_node->value = 20;

new_node->next = first;
```



10

Insert another node (20) – How?

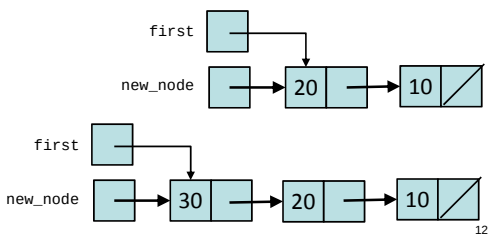
```
new_node = malloc(sizeof(struct node));
new_node->value = 20;

new_node->next = first;
first = new_node;
```



11

Insert another node (30) – How?

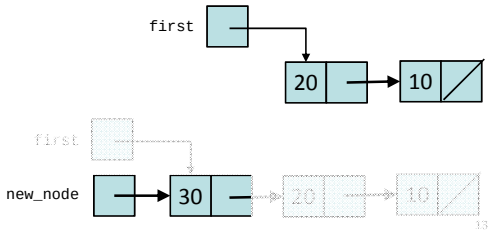
```
new_node = malloc(sizeof(struct node));
new_node->value = 30;
```



12

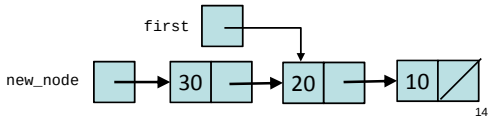
Insert another node (30) – How?

```
new_node = malloc(sizeof(struct node));
new_node->value = 30;
```



Insert another node (30) – How?

```
new_node = malloc(sizeof(struct node));
new_node->value = 30;

new_node->next = first;
first = new_node;
```



14

Insert a node at the Beginning of a list

```
struct node
{
    int value;
    struct node *next;
};

struct node *add_node(struct node *list, int n)
{
    struct node *new_node;
    new_node = malloc(sizeof(struct node));
    new_node->value = n;
    new_node->next = list;
    return new_node;
}
```

15

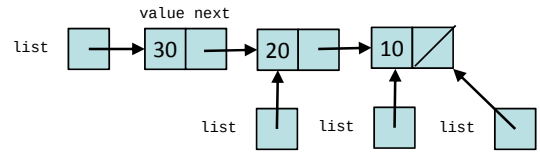
Linked list operations

Beside add_node( ), other possible list operations:

```
struct node *search_node(struct node *list, int n);
void print_list(struct node *list);
struct node* delete_node(struct node *list, int n);
struct node* sort_list(struct node *list);
int sum_list(struct node *list);
…
```

16

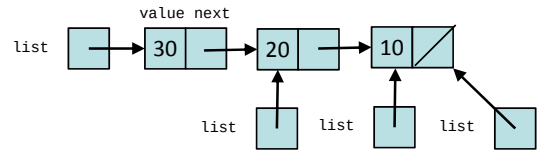
struct node *search_node(struct node *list, int n);




```
for (;;) {
  if (list->value == n)
      return list;
  else
      list = list->next;
  // what if list == NULL?
}
```

17

struct node *search_node(struct node *list, int n);




```
{
  while (list != NULL && list->value != n)
       list = list->next;
    return list;
}
```

18

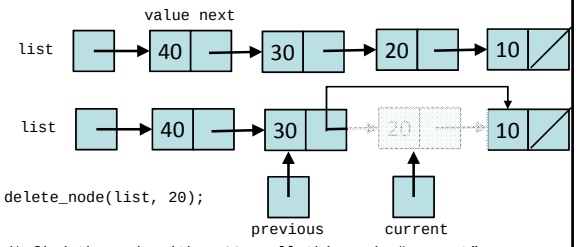
In-Class Exercise 11.1

Copy and paste the skeleton code from the exercise. Implement the `printList()` function.

```
void printList(struct node* head);
```

19

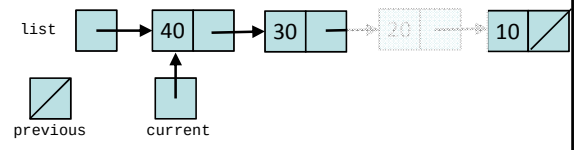
`struct node *delete_node(struct node *list, int n);`




```
/* find the node with n=20, call this node "current"
   find the "previous" node and change its next pointer to
      current->next (what if no previous node?)
    call free() to free the memory of the deleted node
*/
```

20

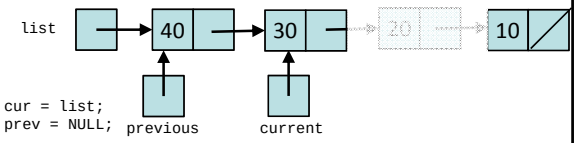
`struct node *delete_node(struct node *list, int n);`




```
cur = list;
prev = NULL;
```

21

struct node *delete_node(struct node *list, int n);

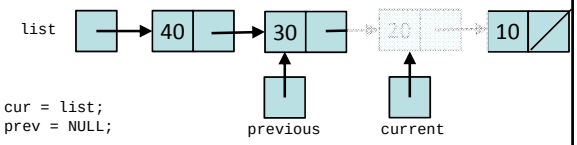



```
cur = list;
prev = NULL;

if (cur->value != n) {
   prev = cur;
   cur = cur->next;
}
```

previous current

22

struct node *delete_node(struct node *list, int n);

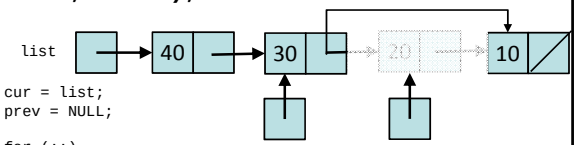



```
cur = list;
prev = NULL;

for (;;)
   if (cur->value != n) {
        prev = cur;
        cur = cur->next;
   }
   else
        break;
}
```

previous current

23

struct node *delete_node(struct node *list, int n);

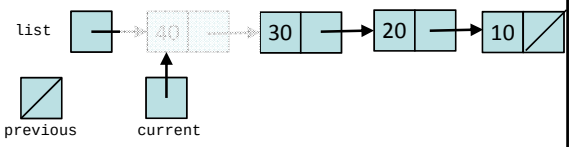



```
cur = list;
prev = NULL;

for (;;)
   if (cur->value != n) {
        prev = cur;
        cur = cur->next;
   }
   else
        break;
}

prev->next = cur->next;
free(cur);
```

previous current

24

struct node *delete_node(struct node *list, int n);




```
delete_node(list, 40); // what if prev == NULL?
```

25

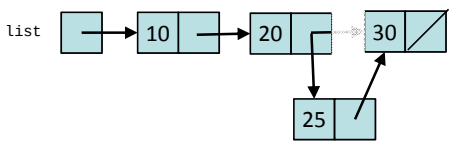
Delete a node

```
struct node *delete_node(struct node *list, int n)
{
    struct node *cur, *prev;
    for (cur = list, prev = NULL;
         cur != NULL && cur->value != n;
         prev = cur, cur = cur->next);

    if (cur == NULL)
        return list;

    if (prev == NULL)
        list = list->next;
    else
        prev->next = cur->next;
    free(cur);
    return list;
}
```

26

In-Class Exercise 11.2

Use your code from Exercise 11.1. Modify `add_node()` such that the list is ordered from small to large.



27

Hexadecimal

- Hexadecimal integer with base of 16.
  - 0-9 represents values 0-9
  - A-F represents values 10-15
- What are these numbers?

```
a = 0x00FF
b = 0xF000
```

28

Bitwise Operations

```
a = 0x00FF
b = 0xF000

a & b = 0x0000 // bitwise and
a | b = 0xF0FF // bitwise or
~a    = 0xFF00 // bitwise not
a ^ b = 0xF0FF // bitwise xor
a << 4 = 0x0FF0 // left shift by 4 bits
a >> 4 = 0x000F // right shift by 4 bits
```

29

Don't Get Confused!

```
x = 0x66, y = 0x93

x & y = 0x02      x && y = 0x01
x | y = 0xF7      x || y = 0x01
~x | ~y = 0xFD    !x || !y = 0x00
x & !y = 0x00     x && ~y = 0x01
```

30

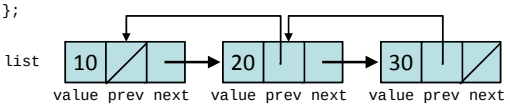
Bitwise Operations

- Bitwise operations are fast.
  - `x * 256` is the same as `x << 8`
  - `x / 256` is the same as `x >> 8`
  - `x % 256` is the same as `(x << 24) >> 24`

31

Assignment 11: Double-Linked List

```
struct node {
   int value;
   struct node *next;
   struct node *prev;
};
```




```
struct node* add_node(struct node *list, int n);  // keep
the list sorted from small to large
struct node* delete_node(struct node *list, int n);
struct node *search_node(struct node *list, int n);
void print_list(struct node *list);
…
```

32