

Linked data structures append and friends

Johan Montelius

KTH

VT26

Cell

```
struct cell {  
    int head;  
    struct cell *tail;  
}
```

```
struct cell *create_cell(int item, struct ceell *tl) {  
    struct cell *new = malloc(sizeof(struct cell));  
    new->head = itm;  
    new->tail = tl;  
}
```

This data structure is also referredd to as a *cons cell*.

Cell

```
struct cell {  
    int head;  
    struct cell *tail;  
}
```

```
struct cell *create_cell(int item, struct ceell *tl) {  
    struct cell *new = malloc(sizeof(struct cell));  
    new->head = itm;  
    new->tail = tl;  
}
```

This data structure is also referedd to as a *cons cell*.

... or

```
struct linked {  
    struct cell *first;  
    int size;  
    :  
}
```

```
struct cell int head; struct cell  
*tail;
```

We have a data structure that represents the empty list.
The data structure will always refer to the first cell of the linked list.

... or

```
struct linked {  
    struct cell *first;  
    int size;  
    :  
}
```

```
struct cell int head; struct cell  
*tail;
```

We have a data structure that represents the empty list.
The data structure will always refer to the first cell of the linked list.

... or

```
struct linked {  
    struct cell *first;  
    int size;  
    :  
}
```

```
struct cell int head; struct cell  
*tail;
```

We have a data structure that represents the empty list.
The data structure will always refer to the first cell of the linked list.

linked list - search

```
bool search(linked *lnk, int key) {  
    struct cell *nxt = lnk->first;  
    while (nxt != NULL) {  
        if (nxt->head == key)  
            return true;  
        nxt = nxt->tail;  
    }  
    return false;  
}
```

linked list - add

```
void add(linked *lnk, int key) {  
    lnk->size += 1;  
    lnk->first = create_cell(key, lnk->first);  
}
```

linke list - delete

```
public void delete(linked *lnk, int key) {  
    struct cell *nxt = lnk->first;  
    struct cell *prev = NULL;  
    while (nxt != NULL) {  
        if (nxt->head == key) {  
            lnk->size -= 1;  
            if (prev != NULL) {  
                prev->tail = nxt->tail;  
            } else {  
                lnk->first = nxt->tail;  
            }  
            break;  
        }  
        prev = nxt;  
        nxt = nxt->tail;  
    }  
}
```

linked list - append

```
void append(linked *a, linked *b) {
    struct cell *nxt = a->first;
    struct cell *prv = NULL;
    while (nxt != NULL) {
        prv = nxt;
        nxt = nxt->tail;
    }
    if (prv != NULL) {
        prv->tail = b->first;
    } else {
        a->first = b->first;
    }
    b->first = NULL;
}
```

linked list - reverse

```
void reverse(struct linked *lnk) {  
  
    struct cell *nxt = lnk->first;  
    struct cell *last = NULL;  
  
    while (nxt != NULL) {  
        struct cell *tmp = nxt;  
        nxt = tmp->tail;  
        tmp->tail = last;  
        last = tmp;  
    }  
    lnk->first = last;  
}
```

sort ...

Let's implement merge sort for a linked list.

sort ...

Let's implement merge sort for a linked list.

we need the length

```
void sort(linked *lnkd) {  
    int n = length(lnkd);  
    lnkd->first = lsort(lnkd->first, n);  
}
```

the recursion

```
struct cell *lsort(struct cell *a, int n) {  
    if( n < 2)  
        return;  
    int na = n/2;  
    int nb = n - na;  
    struct cell *b = lsplitt(a, na);  
    struct cell *sorted_a = lsort(a, na);  
    struct cell *sorted_b = lsort(b, nb);  
    return lmerge(sorted_a, sorted_b);  
}
```

the split

```
struct cell *lsplit(struct cell *a, int n) {  
    cell *last = a->first;  
    for(int i = 1; i < n; i++) {  
        last = last->tail;  
    }  
    struct cell *b = last->tail;  
    last->tail = NULL;  
    return b;  
}
```

the merge easy case

```
static struct cell *lmerge(struct cell *a, struct cell *b)

    if (a == NULL) {
        return b;
    }
    if (b == NULL) {
        return a;
    }
    :
```

.... pick the first, now we have a last cell

```
:
cell *i = a;
cell *j = b;
cell *last = NULL;

if (i->value < j->value) {
    last = i;
    i = i->tail;
} else {
    last = j;
    j = j->tail;
}
struct cell *merged = last;
:
```

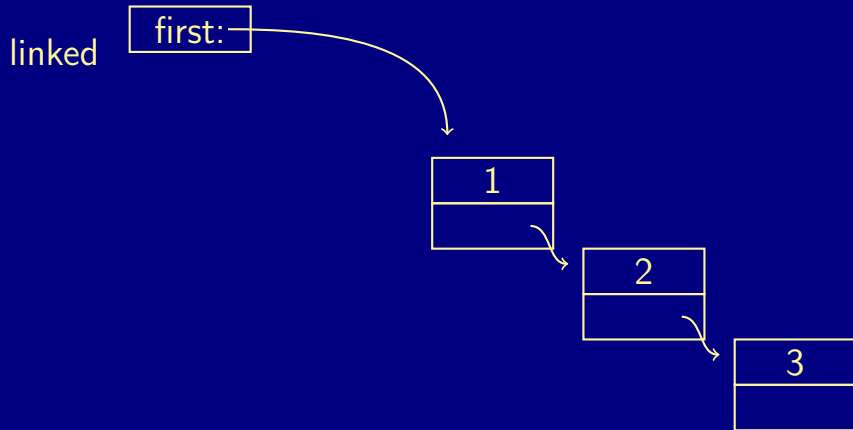
... run through all cells and add them to the last ...

```

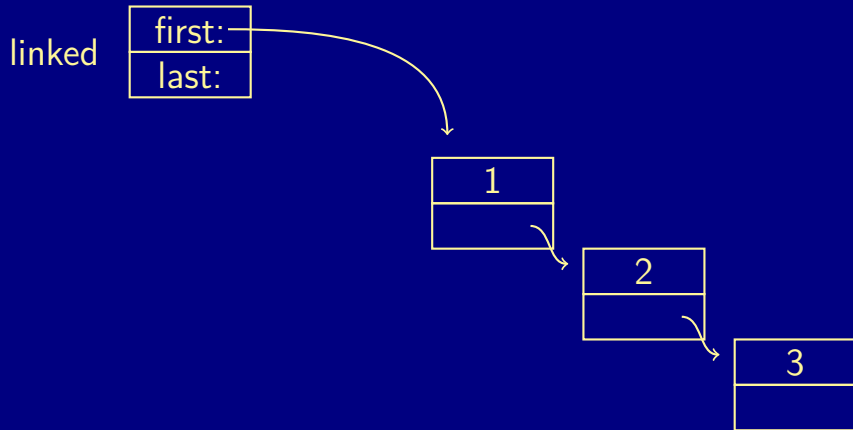
:
while (true) {
    if (i == NULL) { last->tail = j; break; }
    if (j == NULL) { last->tail = i; break; }
    if (i->value <= j->value) {
        last->tail = i;
        last = i;
        i = i->tail;
    } else {
        last->tail = j;
        last = j;
        j = j->tail;
    }
}
return merged;
}

```

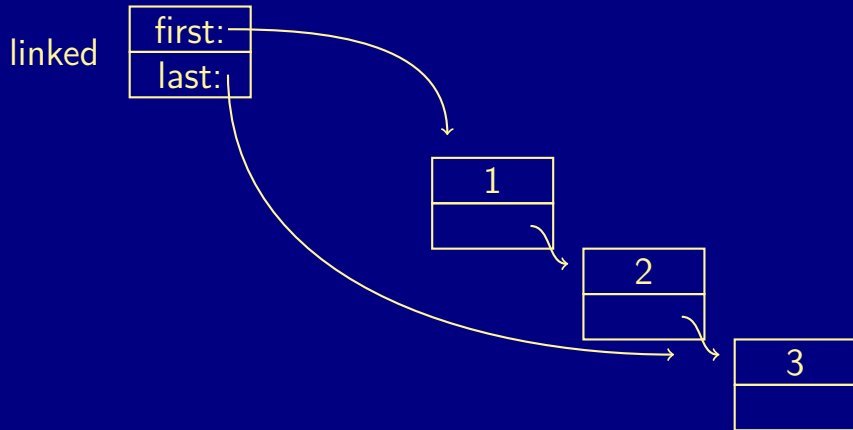
an alternative representation



an alternative representation



an alternative representation



things become simpler

```
void append(linked *a, linked *b) {  
    if (a->last != NULL) {  
        a->last->tail = b->first;  
    } else {  
        this->first = b->first;  
    }  
    a->last = b->last;  
    b->first = NULL;  
    b->last = NULL;  
}
```