CS 201
Dynamic Data Structures (2)

Debzani Deb

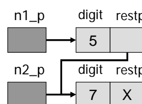
Overview

- Linked list basics
- List Searching
- Insertion
- Deletion
- Stack
- Queue

Linked List basics

- A linked list is a sequence of nodes in which each node but the last contains the address of the next node.

```
typedef struct list_node_s {
    int digit;
    struct list_node_s *restp;
} list_node_t;
list_node_t *n1_p, *n2_p;
n1_p = (list_node_t *) malloc (sizeof(list_node_t));
n2_p = (list_node_t *) malloc (sizeof(list_node_t));
n1_p -> digit = 5;
n2_p -> digit = 7;
n1_p -> restp = n2_p;
n2_p -> restp = NULL;
```



Link List Operation (Searching)

Find First occurrence of target in the list.

```
1.  /*
2.   *  Searches a list for a specified target value.  Returns a pointer to
3.   *  the first node containing target if found.  Otherwise returns NULL.
4.   */
5.  list_node_t *
6.  search(list_node_t *headp,   /* input - pointer to head of list  */
7.         int         target)   /* input - value to search for      */
8.  {
9.       list_node_t *cur_nodep;  /* pointer to node currently being checked */
10.
11.      for  (cur_nodep = headp;
12.            cur_nodep != NULL  &&  cur_nodep->digit != target;
13.            cur_nodep = cur_nodep->restp) {}
14.
15.       return (cur_nodep);
16. }
```

1. What if I put cur_nodep++ instead of cur_nodep -> restp? Could that work? When?
2. What if the order of the following tests are reversed?
   (cur_nodep != NULL) && (cur_nodep -> digit != target)

Link List Operation (Insertion at Head)

- Insert at list's head (i.e. at the front of the list)

```
list_node_t * insertH (list_node_t *pHead, int v) {
        list_node_t *newp;
        newp = (list_node_t *) malloc(sizeof(list_node_t));
        newp->digit = v;
        newp->restp = pHead;
        return newp;  /* return pointer to the new head of the list */
}
```

```
int main(void){
        list_node_t * pHead = NULL;
        pHead = insertH(pHead, 3);
        pHead = insertH(pHead, 5);
        pHead = insertH(pHead, 7);
        pHead = insertH(pHead, 9);
}
```

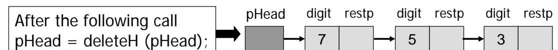
```
typedef struct list_node_s {
   int digit;
   struct list_node_s *restp;
} list_node_t;
```


Link List Operation (Deletion at Head)

- After the insertion at last slide, your list now looks like

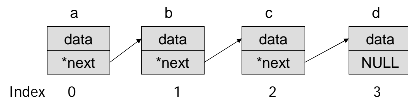
```
list_node_t * deleteH (list_node_t * pHead){
        list_node_t *newp = pHead -> restp;
        /* newp is now pointing to the 2nd element of the list */
        free(pHead);
        return newp;
        /* return pointer to the new head of the list */
}
```

After the following call
pHead = deleteH (pHead);



Lists are arrays

- Quite often the lists are treated as arrays, that can change their size dynamically.



- Assumptions
  - Indexing of list starts at 0 (as in arrays).
  - Every index value is unique.
  - Indices are in growing order (incremented by 1).

Link List Operation (Deletion at Index)

- Delete element at some index of the list

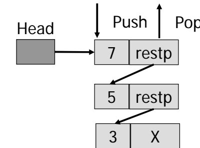
```
list_node_t * deleteIndex (list_node_t * pHead, int index) {
   int i;
   list_node_t * newp, *p, * tmpp;
   if (index == 0) {            // deleting head
        newp = pHead -> restp;
        free(pHead);
        return newp;
   }
```


Link List Operation (Deletion at Index)

```
else {                              // deleting element other than the head
        for (p = pHead, i = 1; (i < index) && (p -> restp != NULL); i++)
        /* searching for the element that has a pointer to the one to be
           deleted */
                p = p -> restp;
        tmpp = p -> restp;
        /* tmpp now points at the element to be deleted */
        if(tmpp != NULL)
                p -> restp = tmpp -> restp;
        /* p now points at the element after the one to be deleted */
        else
        /* we are dealing with the last element of the list */
                p -> restp = NULL;
        free(tmpp);
        return pHead;
   }
}
```


Representing a stack with a linked list

- Like a push down stack of books.
- Push (insert) at top (pointed by Head)
- Pop (remove) from the top (pointed by Head)
- Last in first out (LIFO) architecture.



Push on Stack

```
typedef struct list_node_s {
    int digit;
    struct list_node_s *restp;
} list_node_t;
```

Return pointer to the new head of the stack

Pointer to the current head of the stack

- Push on top of the stack

```
list_node_t * push (list_node_t * sHead, int v) {
   list_node_t * p = (list_node_t *)
   malloc(sizeof(list_node_t));
   p -> digit = v;
   p -> restp = sHead;
   return p;
}
```

Return the new node as the head of the stack

```
int main(void) {
  list_node_t * sHead = NULL;
  /* Function call for push*/
  sHead = push(sHead, 3);
  sHead = push(sHead, 5);
  return 0;
}
```


Pop from Stack

Return pointer to the new head of the stack

Pointer to the current head of the stack

Popped digit as output parameter

- Pop from the top of the stack

```
list_node_t * pop (list_node_t * sHead, int * v) {
   list_node_t * p;
   *v = sHead ->digit;
   p = sHead -> restp;
   free (sHead);
   return p;
}
```

Return the new node as the head of the stack

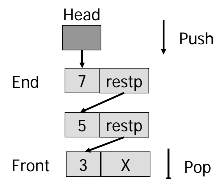
```
int main(void) {
  list_node_t * sHead = NULL;
  int val;
  sHead = push(sHead, 3);
  sHead = push(sHead, 5);
 /* Function call for pop */
  sHead = pop (sHead, &val);
  printf("Popped : %d", val);
  return 0;
}
```

Stack (Summary)

- In stack only top element can be accessed.
- You could make a stack with an array.
  - Linked list is just one way.
- Common design for function invocation.
- Both push and pop are constant time operations on stack.

Representing a Queue with a linked list

- Like a queue of people waiting
- Push at the Head (i.e at the end of the list).
- Pop from the Bottom (i.e from the front of the list)
- First In First Out (FIFO)



Push on Queue

```
typedef struct list_node_s {
   int digit;
   struct list_node_s *restp;
} list_node_t;
```

Return pointer to the new head of the queue

Pointer to the current head of the queue

- Push is same as stack (at Head)

```
list_node_t * push (list_node_t * qHead, int v) {
   list_node_t * p = (list_node_t *)
   malloc(sizeof(list_node_t));
   p -> digit = v;
   p -> restp = qHead;
   return p;
}
```

Return the new node as the head of the queue

```
int main(void) {
  list_node_t * qHead = NULL;
  /* Function call for push*/
  qHead = push(qHead, 3);
  qHead = push(qHead, 5);
  return 0;
}
```


Pop from Queue (from the bottom)

```
list_node_t * pop (list_node_t * qHead, int * v) {
   list_node_t * qEnd, * qFront = NULL;
   if (qHead -> restp = NULL) {  // Queue has only one element
     *v = qHead ->digit;
     free (qHead);
     return NULL;
   }
   for (qEnd = qHead; qEnd ->restp != NULL; qEnd = qEnd -> restp)
      qFront = qEnd;
   *v = qEnd -> digit;
   qFront -> restp = NULL;
   free(qEnd);
   return qHead;
}
```

Can we write this more efficiently?

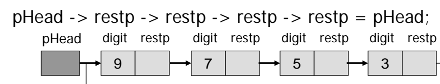
Queue (Summary)

- You could implement it as an array too.
- You could make a hybrid of stack/queue to access at either end.
- Common design for process scheduling, event processing, buffering, input/output etc.
- In our design push is constant time, but pop is O(n) linear time (where n is the number of elements in the queue).
- If we record two pointers (front and end) instead of only one pointer pointing to the head of the list – both push and pop would have constant time.
  - See the implementation in your textbook.

Circular and double linked list

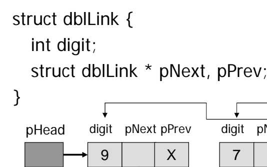
- Circular linked list

  pHead -> restp -> restp -> restp -> restp = pHead;



- Double linked list

```
struct dblLink {
   int digit;
   struct dblLink * pNext, pPrev;
}
```



Important !

- Lab 6 grades are posted.
- Remember Exam 2 next Wednesday.
- Covers
  - Array
  - String
  - Recursion
  - Structure & Union
  - Dynamic data storage (upto queue)
- All rules of exam 1 will be applicable.