Singly Linked List Worksheet

A visual and practical guide to sentinel nodes, insertion, reversal, and linked-list operations

Prepared for classroom lab use

Section 1: Introduction

This worksheet explores a basic singly linked list implementation in C. It covers node creation, insertion, traversal, sentinel head nodes, list reversal (with and without sentinel), and several exercises for practice. The provided code files are: **main.c**, **listlib.c**, and **Makefile**.

Section 2: Code Listings

listlib.c

```
#include "listlib.h"
void inputlst(NodePtr hd)
{
  int c;
  do
  {
    scanf("%d", &c);
    insert(hd,c);
  }while(c != -1);
}
void insert(NodePtr l, int val)
{
  NodePtr p=(NodePtr)malloc(sizeof(Node));
  p->val = val;
  p->next = NULL;
  while(l->next !=NULL)
  {
    l = l->next;
  }
  l->next=p;
}

void printlst(NodePtr l)
{
  while(l)
  {
    printf("%d\n", l->val);
    l=l->next;
  }
}
```

main.c

```c
#include "listlib.h"
void main()
{
  NodePtr hd= (NodePtr) malloc(sizeof(Node));
  hd->val =-1;
  hd->next = NULL;

  inputlst(hd);
  printlst(hd);
}
```

Makefile

```
# Makefile

# target program

# default rule
a.out : main.o listlib.o
■   gcc main.o listlib.o

# compile source files into object files
main.o: main.c listlib.h
■gcc -c main.c

listlib.o: listlib.c listlib.h
■   gcc -c listlib.c

# clean up build files
clean:
■rm -f *.o a.out
```

Section 3: Function Walkthroughs

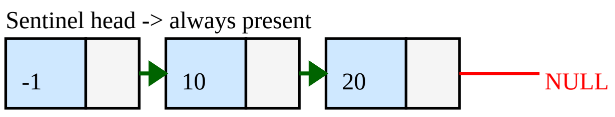
inputlst(NodePtr hd)

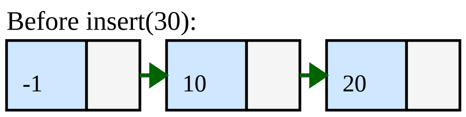
Reads integers from stdin and inserts them into the list by calling insert(). The function uses the sentinel head node 'hd' as the fixed starting point; actual data nodes are in hd->next.

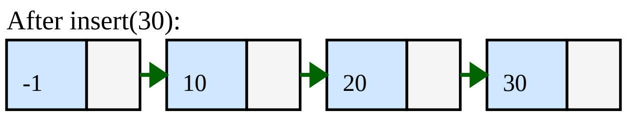
insert(NodePtr l, int val)

Creates a new node with the given value and walks to the end of the list starting from 'l'. Because 'l' is often the sentinel head, the algorithm is a simple loop that appends the new node at the end (last->next = new_node). Time complexity: O(n) per insert (if inserting at tail using traversal).

Insert: Visual example







printlst(NodePtr l)

Traverses the list starting at l->next (if l is sentinel) or at l (if non-sentinel), printing node values until NULL. Traversal uses a temporary pointer that moves along next pointers.

Section 4: Sentinel Head Nodes

A sentinel head node is a dummy node placed at the beginning of the list. It simplifies many operations because the head pointer itself is never NULL; only head->next can be NULL. Benefits: - Avoids special-case handling for empty lists when inserting or deleting. - Simplifies deletion near the head (no separate 'if head==NULL' case). - Makes traversal code uniform (start at head->next).

No sentinel: empty list = NULL

Code comparison:

```
// With sentinel
NodePtr hd = malloc(sizeof(Node));
hd->val = -1; // sentinel value
hd->next = NULL;
insert(hd, 10); // no special-case needed
// Without sentinel
NodePtr head = NULL;
insert_without_sentinel(&head, 10); // must handle head==NULL inside
```

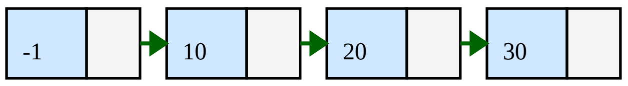
Section 5: Reversing the List

Reverse with sentinel

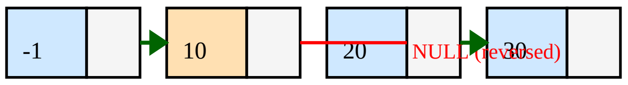
```
void reverse(NodePtr hd) {
    NodePtr prev = NULL, curr = hd->next, next = NULL;
    while (curr != NULL) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    hd->next = prev;
}
```

Explanation: We keep 'hd' fixed and rewire hd->next to the new head (prev) after reversing. Safe when list empty (hd->next == NULL).

Reverse steps (illustrative):



Step 0



Step 1: first node reversed (10->NULL)

Reverse without sentinel

```
void reverse(NodePtr *head) {
    NodePtr prev = NULL, curr = *head, next = NULL;
    while (curr != NULL) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    *head = prev;
}
```

Note: Must ensure *head is valid (may be NULL) and caller updates the head pointer correctly.

Section 6: Exercises

Exercise 1 — Count elements Write and test a function that returns the number of data nodes in the list. Try both versions: starting from hd and hd->next. Explain the difference.

Exercise 2 — Find maximum element Write a function to find and return the maximum value in the list. How does sentinel remove the need for special-case checks?

Exercise 3 — Delete an element with key Implement delete(hd, key) that removes the first node whose value equals key. Trace the behavior when deleting at start, middle, and end.

Section 7: Summary and Build Instructions

Summary: Sentinel nodes simplify linked-list code by guaranteeing a non-NULL head. Reversal can be done safely in-place by rewiring next pointers. Exercises encourage students to trace and implement common operations. Build instructions (Makefile): - Run `make` to compile (produces a.out). - Run `./a.out` to execute program. - Run `make clean` to remove binaries and object files.

Appendix: Visual Key

Blue box: node; Green arrow: pointer next; Red text/line: NULL or important pointer update