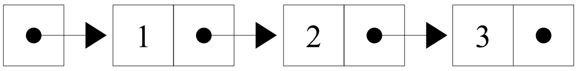
CMSC 21 Handout: Linked List Visualization

updated by ajjacildo

Linked List

The primary use of a linked list is to form a flexible data storage structure whose size can "expand or shrink" to accommodate varying sizes of data. A linked list can be viewed as a chain of nodes. Each node has two components, a data component and a link component. The link component of a node is a link to another node, this allows the chaining of nodes in a linked list. Nodes in a linked list are created and deleted dynamically or "on demand". This makes a linked list more flexible compared to an array. Here's an illustration of a linked list of integers.

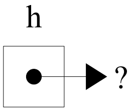


A linked list node in C is implemented as a self-referential structure. A self-referential structure is a structure that contains a *pointer field* that can point to a structure similar to itself. For example, we define a self-referential structure `struct node_tag`.

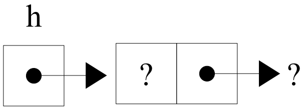
```
struct node_tag{
      int x;
      struct node_tag *next;
};
```

The structure contains two fields: `x` (data component) and `next` (link component), where `x` is an integer while `next` is a pointer to a `struct node_tag`. The `next` field makes `struct node_tag` a self-referential structure. To discuss how to build a linked list, consider the sample program below with the corresponding visualization.

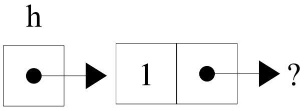
```
//Sample code for linked list
#include<stdio.h>

typedef struct nodetag{
  int x;
  struct nodetag *next;
}node;

int main(){
  node *h, *temp;
```



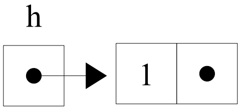
```
  //first node
  h=(node *)malloc(sizeof(node));
```



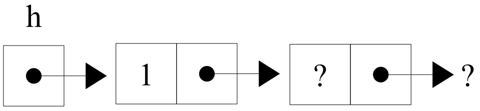
```
  h->x=1;
```



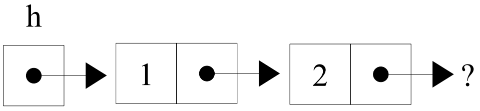
```
  h->next=NULL;
```



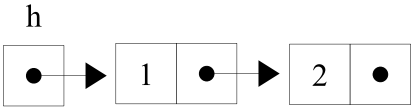
```
  //second node
  h->next=(node *)malloc(sizeof(node));
```



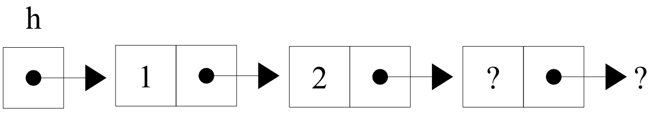
```
  h->next->x=2;
```



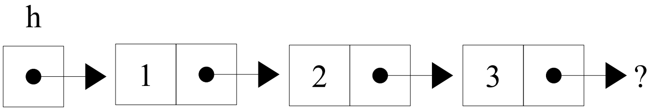
```
  h->next->next=NULL;
```



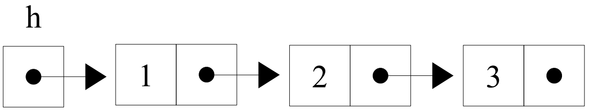
```
//third node
h->next->next=(node *)malloc(sizeof(node));
```



```
h->next->next->x=3;
```

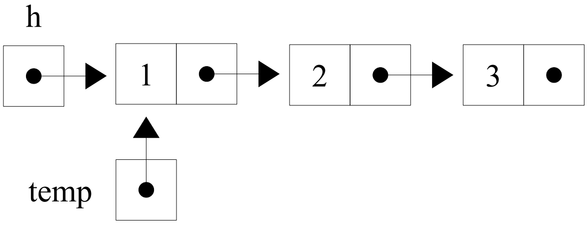


```
h->next->next->next=NULL;
```

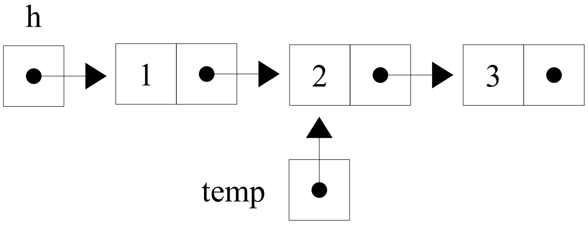


```
//display the contents of the linked list

temp=h;
while(temp!=NULL){
    printf("%3i ",temp->x);
    temp=temp->next;
}
printf("\n");
```

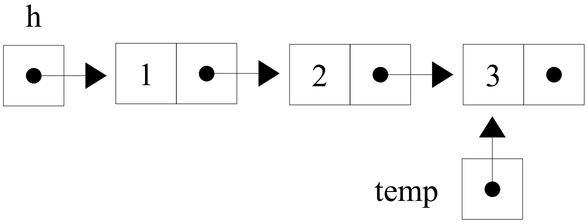
step 0: `temp=h;`



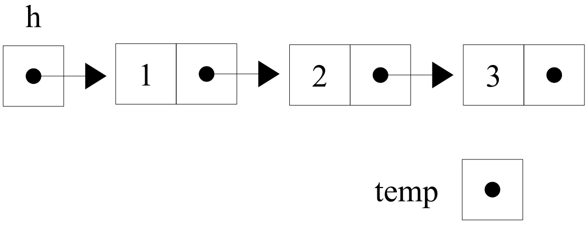
step 1: after (1st) `temp=temp->next;`



step 2: after (2nd) `temp=temp->next;`

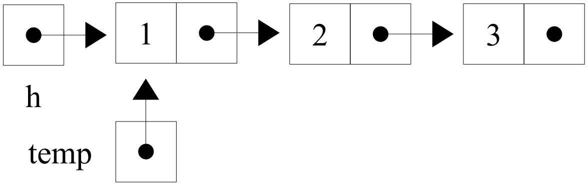


step 3:after (3rd) `temp = temp->next;`

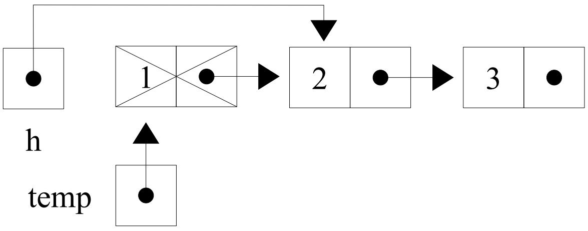


```
  //deallocation

  while(h!=NULL){
      temp=h;
      h=h->next;
      free(temp);
  }

  return(0);

}//end of main
```

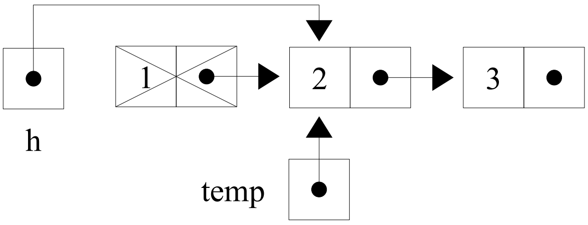
step 1a: `after (1st) temp=h;`



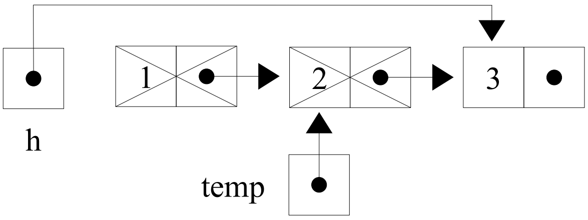
step 1b: `after (1st) h=h->next;`
`free(temp);`



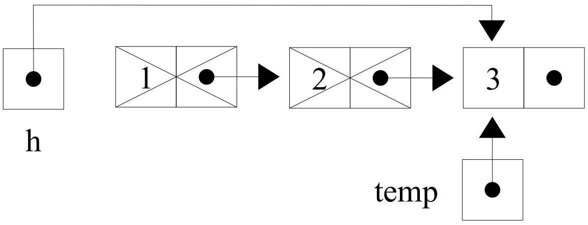
step 2a: `after (2nd) temp=h;`



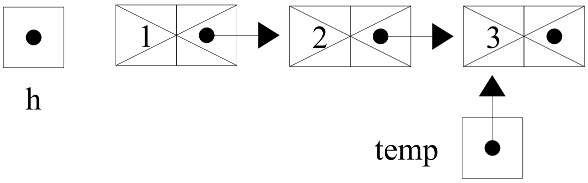
step 2b: `after (2nd) h=h->next;`
`free(temp);`



step 3a: `after (3rd) temp=h;`

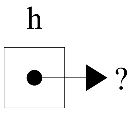


step 3b: `after (3rd) h=h->next;`
`free(temp);`

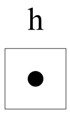


Drawing conventions:

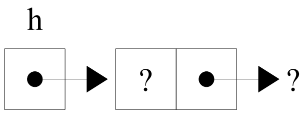
```
//uninitialized pointer
node *h;
```



```
//null pointer
node *h=NULL;
```



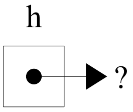
```
//malloc
h=(node *)malloc(sizeof(node));
```



Pen-and-Paper Exercise

Now try it on your own and submit your answers to your lab instructor. Given the drawing conventions discussed earlier, draw and the effect of each of the assignment statements in the given sample code.

```
//Sample code for linked list
#include<stdio.h>

typedef struct nodetag{
  int x;
  struct nodetag *next;
}node;

int main(){
  node *h, *temp;
```



```
  //first node
  h=(node *)malloc(sizeof(node));
```

____

```
  h->x=1;
```

____

```
  h->next=NULL;
```

____

```
  //second node
  h->next=(node *)malloc(sizeof(node));
```

____

```
  h->next->x=2;
```

____

```
  h->next->next=NULL;
```

____

```
  //third node
  h->next->next=(node *)malloc(sizeof(node));
```

____

```
  h->next->next->x=3;
```

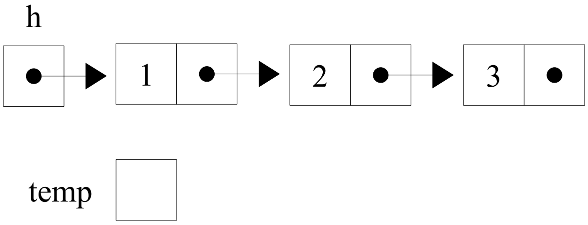
____

```
  h->next->next->next=NULL;
```

____

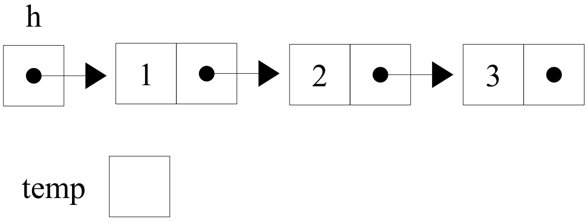
```
  //display the contents of the linked list

  temp=h;
  while(temp!=NULL){
      printf("%3i ",temp->x);
      temp=temp->next;
  }
  print("\n");
```

step 0: temp=h;

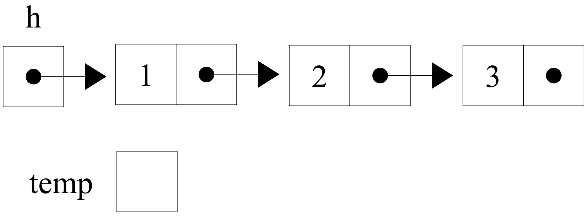


```
  //deallocation

  while(h!=NULL){
      temp=h;
      h=h->next;
      free(temp);
  }

  return(0);

}//end of main
```

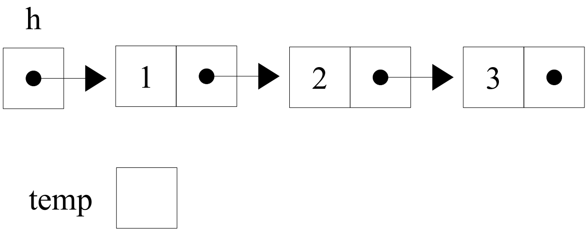
step 1: after (1st) temp=temp->next;



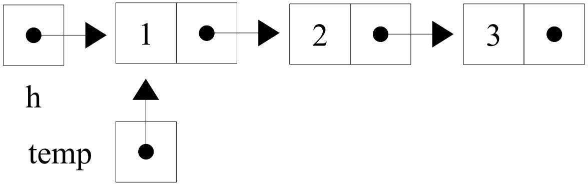
step 2: after (2nd) temp=temp->next;



step 3:after (3rd) temp = temp->next;



step 1a: after (1st) temp=h;



step 1b: after (1st) h=h->next;
free(temp);

step 2a: after (2nd) temp=h;

step 2b: after (2nd) `h=h->next;`
`free(temp);`

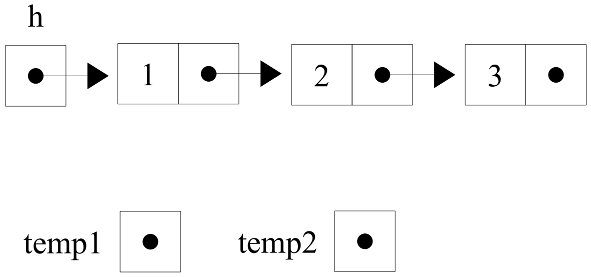
step 3a: after (3rd) `temp=h;`

step 3b: after (3rd) `h=h->next;`
`free(temp);`

Now consider the code snippet below and trace it starting from the given initial setup. Go through the loop and draw the effect of **EACH** of the assignment, starting from the first **temp1=h**; assignment statement up to the last **temp2=temp1**; assignment statement and finally, the effect of **h=temp2**; assignment statement.

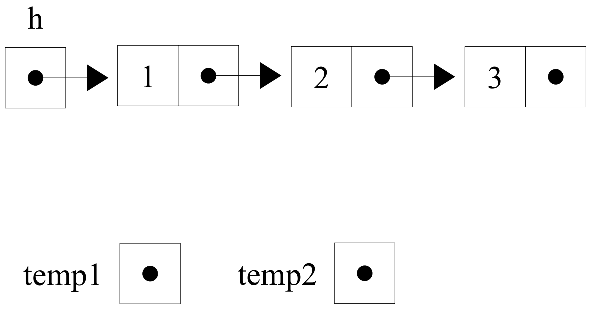
Hint: For this part you are encouraged to do this one drawing per page so it can be browsed like a "flip book".

```
//assume node *h; holds the initial list

node *temp1=NULL, *temp2=NULL;

while (h!=NULL){
      temp1=h;
      h=h->next;
      temp1->next=temp2;
      temp2=temp1;
}
h=temp2;
```



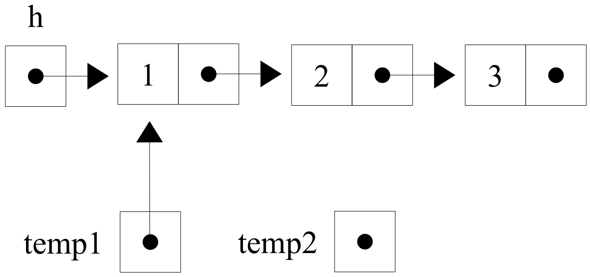
INITIAL SETUP

```
//assume node *h; holds the initial list

node *temp1=NULL, *temp2=NULL;

while (h!=NULL){
      temp1=h;
      h=h->next;
      temp1->next=temp2;
      temp2=temp1;
}
h=temp2;
```



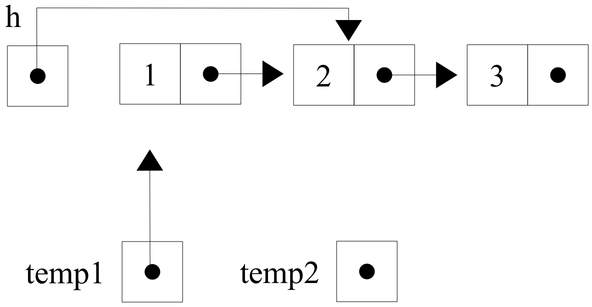
AFTER 1st temp1=h;

```
//assume node *h; holds the initial list

node *temp1=NULL, *temp2=NULL;

while (h!=NULL){
      temp1=h;
      h=h->next;
      temp1->next=temp2;
      temp2=temp1;
}
h=temp2;
```



AFTER 1st h=h->next;

```
//assume node *h; holds the initial list

node *temp1=NULL, *temp2=NULL;

while (h!=NULL){
       temp1=h;
       h=h->next;
       temp1->next=temp2;
       temp2=temp1;
}
h=temp2;
```


h
1
2
3
temp1
temp2