2016

International Journal of Humanities & Information Technology (IJHIT)

Volume – 1, Issue – 2

April 2016 Issue

ISSN : 2456 - 1142

INTERNATIONAL JOURNAL OF HUMANITIES & INFORMATION TECHNOLOGY (IJHIT)

WWW.IJHIT.COM

International Journal of Humanities & Information Technology (IJHIT)
*e-ISSN: **2456 - 1142**, Volume 1, Issue 2, (April 2016), www.ijhit.com*

Fundamental of Singly Linked List Creation and Insertion and Analysis

Author

Sanjay S. Reddy[1], Saddanand Rao[2], Peruwam Thomas[3]

[1,2](Research Scholar/ Department of Computer Science/Chhattisgarh Swami Vivekanand Technical University, Bjilai, Chhattisgarh/India)
[3](Assistant Professor/ Department of Computer Science/Chhattisgarh Swami Vivekanand Technical University, Bjilai, Chhattisgarh/India)

Abstract

Linked list is a dynamic data structure that can be grow and shrink at run time. insertion and deletion operation are easier. Efficient memory utilization there is no need to pre memory allocation. Faster access time can be expanded in constant access time without memory overhead. linear data structure is easy to implement such like stack and queue.

Keyword: Malloc, Calloc, SLL, pointer, new_node, Current_node, Start

1. Introduction

Linked list creation proceed following step in creation

***Step 1 :* Include Alloc.h Header File**

We don't know, how many nodes user is going to create once he execute the program. In this case we are going to allocate memory using Dynamic Memory Allocation functions such as Alloc & Malloc. Dynamic memory allocation functions are included in alloc.h
#include<alloc.h>

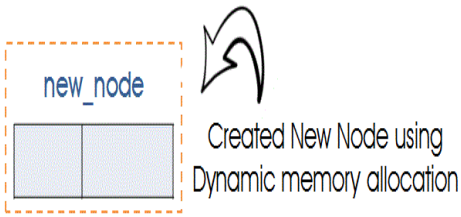
Step 2 : Define Node Structure

We are now defining the new global node which can be accessible through any of the function.

```
struct node {
  int data;
  struct node *next;
}*start=NULL;
```

Step 3 : Create Node using Dynamic Memory Allocation

Now we are creating one node dynamically using malloc function.We don't have prior knowledge about number of nodes, so we are calling malloc function to create node at run time.

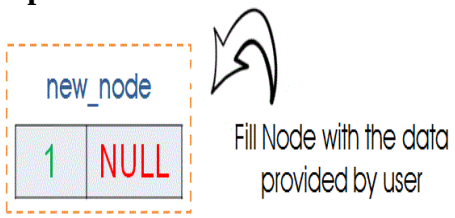



```
new_node=(struct node *) malloc (sizeof (struct node));
```

Step 4 : Fill Information in newly Created Node

Now we are accepting value from the user using scanf. Accepted Integer value is stored in the data field.

Tip #1 Whenever we create new node, Make its Next Field as NULL.

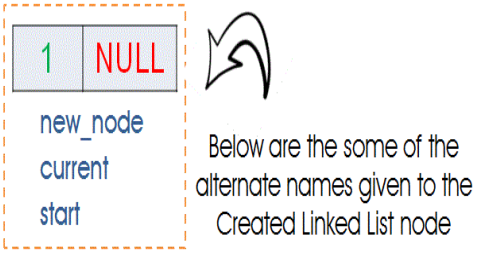



```
printf("nEnter the data : ");
scanf("%d",&new_node->data);
new_node->next=NULL;
```

Step 5 : Creating Very First Node

If node created in the above step is very first node then we need to assign it as Starting node. If start is equal to null then we can identify node as first node –
start == NULL

First node has 3 names : new_node,current,start




```
if(start == NULL)
{
  start = new_node;
  curr = new_node;
}
```

Step 6 : Creating Second or nth node

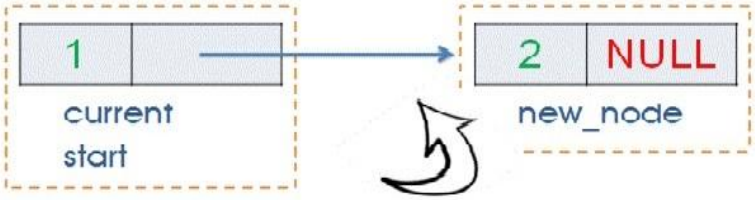
Lets assume we have 1 node already created i.e we have first node. First node can be referred as "new_node","curr","start".
Now we have called create() function again

Now we already have starting node so control will be in the else block –

```
else
   {
   current->next = new_node;
   current = new_node;
   }
```

Inside Else following things will happen –

In the else block we are making link between new_node and current node.



Making Link between Current and new node

current->next = new_node;

Now move current pointer to next node –

current = new_node;

Some Terminology Of Linked List :

Consider the following node structure –

```
struct node
 {
    int data;
    struct node *next;
}*start = NULL;
```

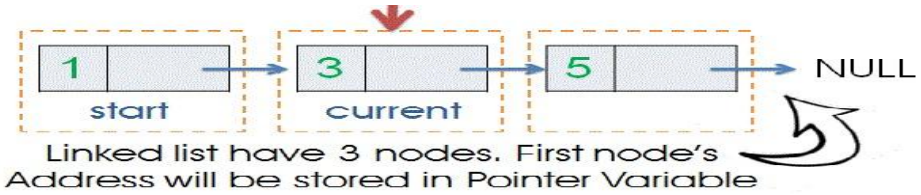
struct node *new_node,*current;

Summary of Different Linked List Terms :

Declaration Term	Explanation
struct node *new_node,*current;	Declaring Variable of Type Node.
*start = NULL;	Declared a global variable of type node. “start” is used to refer the starting node of the linked list.
start->next	Access the 2nd Node of the linked list. This term contain the address of 2nd node in the linked list. If 2nd node is not present then it will return NULL.
start->data	It will access “data” field of starting node.
start->next->data	It will access the “data” field of 2nd node.

current = start->next	Variable "current" will contain the address of 2nd node of the linked list.
start = start->next;	After the execution of this statement, 2nd node of the linked list will be called as "starting node" of the linked list.

Explanation of Terms :



Consider the above linked list , we have initial conditions –

1. "start" node is pointing to First Node [data = 1]
2. "current" node is pointing to Second Node [data = 3]

with respect to the above linked list , data will be –

Term	Explanation
current = start->next	"current" will point to node having data = 3
int num = start->data	"num" will contain integer value "1".
int num = start->next->data	"num" will contain integer value "3".
temp = current->next	"temp" will point to node having data = 5
int num = current->data	"num" will contain integer value "3".
int num = current->next->data	"num" will contain integer value "5".

Position of Insertion in SSL:

There are the position to insert node in SSL

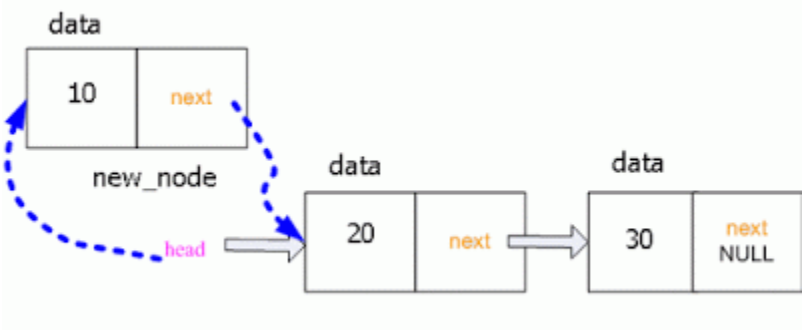
a) Inserting node at start in the Singly Linked List
b) Insert node at Last / End Position in Singly Linked List
c) Insert Node at Middle Position in Singly Linked List

A) Inserting Node at Start in The SLL (Steps):

1. **Create** New Node
2. Fill Data into "**Data Field**"
3. Make it's "**Pointer**" or "**Next Field**" as **NULL**
4. Attach This newly **Created node to Start**
5. Make newnode as **Starting node**

```
void insert_at_beg()
{
struct node *new_node,*current;

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

 if(new_node == NULL)
    printf("nFailed to Allocate Memory");

 printf("nEnter the data : ");
 scanf("%d",&new_node->data);
 new_node->next=NULL;

   if(start==NULL)
   {
   start=new_node;
   current=new_node;
   }
   else
   {
   new_node->next=start;
   start=new_node;
   }
}
```

Diagram



Attention

1. If starting node is not available then **"Start = NULL"** then following part is executed

```
if(start==NULL)
      {
      start=new_node;
      current=new_node;
      }
```

2. If we have previously created First or starting node then **"else part"** will be executed to insert node at start

```
else
      {
      new_node->next=start;
      start=new_node;
      }
```

b) Insert node at Last / End Position in Singly Linked List

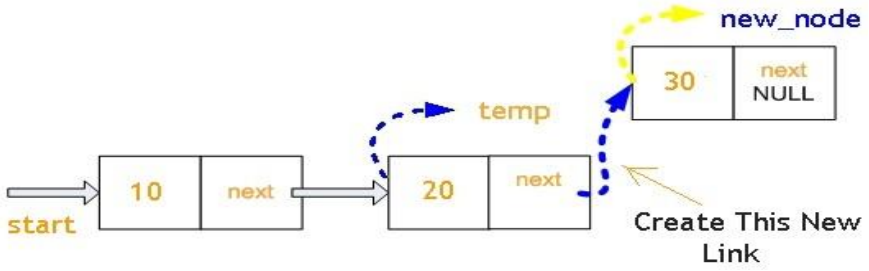
Inserting node at start in the SLL (Steps):

1. **Create** New Node
2. Fill Data into "**Data Field**"
3. Make it's "**Pointer**" or "**Next Field**" as **NULL**
4. Node is to be inserted at Last Position so we need to traverse **SLL upto Last Node**.
5. Make link between **last node and new node**

```
void insert_at_end()
{
struct node *new_node,*current;

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

if(new_node == NULL)
   printf("nFailed to Allocate Memory");

 printf("nEnter the data : ");
 scanf("%d",&new_node->data);
 new_node->next=NULL;

 if(start==NULL)
 {
   start=new_node;
   current=new_node;
```

```
 }
 else
 {
   temp = start;
     while(temp->next!=NULL)
     {
     temp = temp->next;
     }
   temp->next = new_node;
 }
}
```

Diagram



Attention

1. If starting node is not available then **"Start = NULL"** then following part is executed

```
if(start==NULL)
      {
      start=new_node;
      current=new_node;
      }
```

2. If we have previously created First or starting node then **"else part"** will be executed to insert node at start
3. Traverse Upto Last Node., So that **temp** can keep track of Last node

```
else
      {
      temp = start;
   while(temp->next!=NULL)
  {
  temp = temp->next;
  }
```

Make **Link between Newly Created node and Last node** (temp)
temp->next = new_node;

To pass Node Variable to Function Write it as –

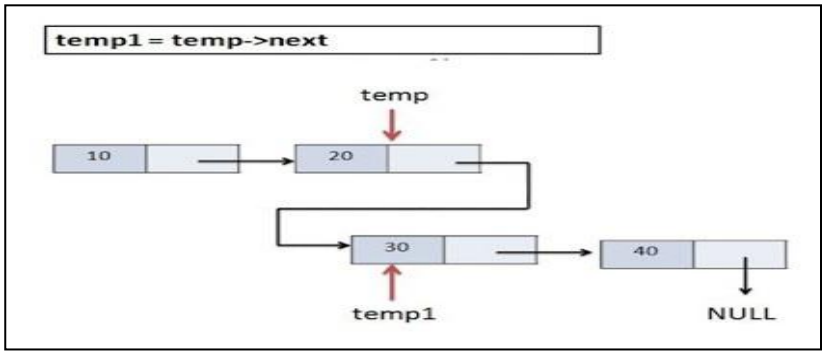
void insert_at_end(struct node *temp)

c) Insert Node at Middle Position in Singly Linked List

```
void insert_mid()
{
   int pos,i;
   struct node *new_node,*current,*temp,*temp1;
   new_node=(struct node *)malloc(sizeof(struct node));
   printf("nEnter the data : ");
   scanf("%d",&new_node->data);
   new_node->next=NULL;
   st :
   printf("nEnter the position : ");
   scanf("%d",&pos);
   if(pos>=(length()+1))
     {
     printf("nError : pos > length ");
     goto st;
     }
   if(start==NULL)
     {
     start=new_node;
     current=new_node;
     }
   else
     {
     temp = start;
          for(i=1;i< pos-1;i++)
          {
          temp = temp->next;
          }
     temp1=temp->next;
     temp->next = new_node;
     new_node->next=temp1;
     }
}
```

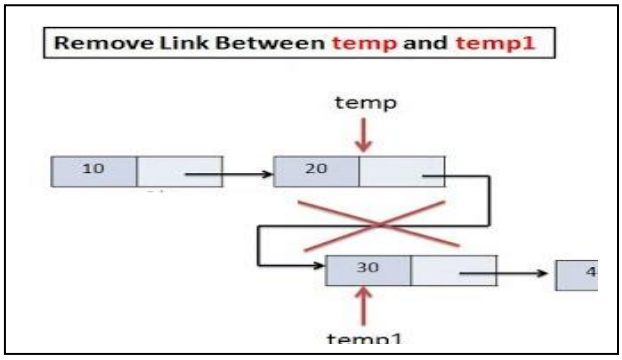
Explanation :

Step 1 : Get Current Position Of "temp" and "temp1" Pointer. = temp->next;



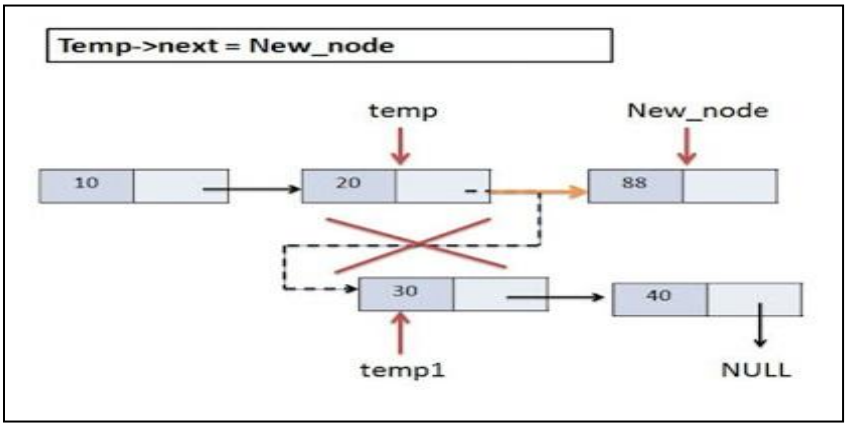

```
temp = start;
        for(i=1;i< pos-1;i++)
 {
 temp = temp->next;
 }
```

Step 2



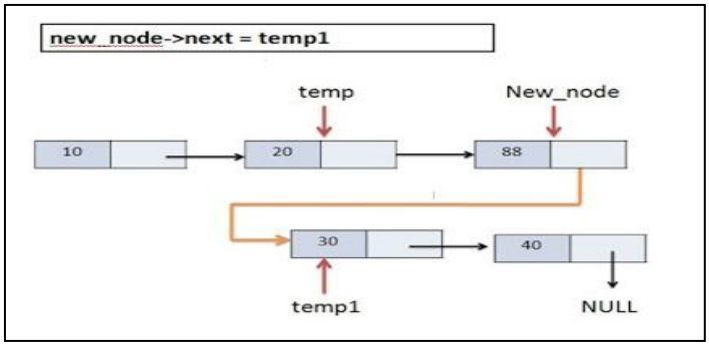

```
temp1=temp->next;
```

Step 3 : _node;




```
temp->next = new_node;
```

Step 4



new_node->next = temp1

References

[1]. *Skiena, Steven S. (2009). The Algorithm Design Manual (2nd ed.). Springer. p. 76. ISBN 9781848000704. We can do nothing without this list predecessor, and so must spend linear time searching for it on a singly-linked list.*
[2]. http://www.osronline.com/article.cfm?article=499
[3]. http://www.cs.dartmouth.edu/~sergey/me/cs/cs108/rootkits/bh-us-04-butler.pdf
[4]. *Chris Okasaki (1995). "Purely Functional Random-Access Lists". Proceedings of the Seventh International Conference on Functional Programming Languages and Computer Architecture: 86–95. doi:10.1145/224164.224187.*
[5]. Gerald Kruse. CS 240 Lecture Notes: Linked Lists Plus: Complexity Trade-offs. Juniata College. Spring 2008.
[6]. *Day 1 Keynote - Bjarne Stroustrup: C++11 Style* at *GoingNative 2012* on *channel9.msdn.com* from minute 45 or foil 44
[7]. *Number crunching: Why you should never, ever, EVER use linked-list in your code again* at *kjellkod.wordpress.com*
[8]. *Brodnik, Andrej; Carlsson, Svante; Sedgewick, Robert; Munro, JI; Demaine, ED (1999), Resizable Arrays in Optimal Time and Space (Technical Report CS-99-09) (PDF), Department of Computer Science, University of Waterloo*