GIZMOFACTS

LEARN DATA STRUCTURE SORTING & FILE HANDLING IN C

Kaustav Ghosh Dastidar

www.gizmofacts.com

Learn Data Structure Sorting & File Handling in C

Introduction

This book covers the fundamental concepts involved in Data Structure Sorting & File Handling in C. We will look at how they are affected by the design of appropriate data structures, as well as how some structures and algorithms are more efficient than others in for the same task. We will focus on a few fundamental tasks that underpin much of computer science, such as stacks and linked lists and sorting programs, but the techniques discussed will be much more general. We will start by looking at some fundamental data structures like arrays, lists, pointers, queues, stacks, and trees, and then look at how they are used in various searching, sorting, and file-handling algorithms in C.

We will investigate the computational efficiency of the algorithms we develop, as well as gain intuitions about the advantages and disadvantages of the various potential approaches for each task.

It is hoped that programmers and students will find this book interesting and useful to all those who want to achieve the desired level of competence, not only in exams but also in their efforts to solve real-world problems. Critical feedback and constructive suggestions for improving the E-book are welcome and greatly appreciated.

With Warm Regards,
Kaustav Ghosh Dastidar
Author & Founder of Gizmofacts

About The Author

The man behind **Gizmofacts, Kaustav Ghosh Dastidar** holds a Master Degree in Computer Science. He is usually turned on (metaphorically) by technology and gadgets. Born and raised in a quintessential middle class family he has been well aware of the ignorance the mass harbours about technology. Through Gizmofacts he wants to reach out to all those people, who he believes just need a little push to get into this unique and amazing world of science and software.

Moreover, Kaustav is well aware that nurturing an interest in gadgets doesn't come cheap. Hence he wants to also be an enabler who would provide all the 'need to know' financial details of different gadgets so that people can live their dreams remaining in their budget.

To know more about tips and tricks of softwares, gizmos and mobile apps, follow him in ***Twitter***, ***Facebook*** and Google+.

You may also subscribe to Gizmofacts in ***Youtube*** for getting more information about software tips & tricks.

Disclaimer

Copyright @ 2022 Kaustav Ghosh Dastidar All rights reserved.

This eBook may not be copied or distributed without permission in any way. This publication's content is offered solely for informational reasons. The usage or misuse of this eBook, as well as any financial damage incurred by individuals or property as a direct or indirect result of using this eBook, are not the author's responsibility.

We are unable to guarantee your success or outcomes in the future due to some unforeseeable risks associated with doing business online. You accept that the author is not responsible for any success or failure of your business that is related in any way to the download and use of our information and that the use of our information should be based on your due diligence.

Without the author's prior written consent, no portion of this eBook may be copied or otherwise distributed in any way, including electronically, mechanically, by photocopy, recording, or any other method

Table of Content

C Program-To Sort Elements Using Bubble Sort

Code description

This is a simple sorting algorithm. In this algorithm each element is compared with ***adjacent element*** and ***swapped*** if their position is incorrect. This algorithm is named as bubble sort because after every pass the largest element moves to the end of the array same as like bubbles, the lighter elements come up and heavier elements settle down.

Example: In each step, elements written in bold are being compared.

First Pass:

(5 1 4 2 8) –> (1 5 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.
(1 5 4 2 8) –> (1 4 5 2 8), Swap since 5 > 4
(1 4 5 2 8) –> (1 4 2 5 8), Swap since 5 > 2
(1 4 2 5 8) –> (1 4 2 5 8), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Second Pass:

(1 4 2 5 8) –> (1 4 2 5 8)
(1 4 2 5 8) –> (1 2 4 5 8), Swap since 4 > 2
(1 2 4 5 8) –> (1 2 4 5 8)
(1 2 4 5 8) –> (1 2 4 5 8)
Now, the array is already sorted, but the algorithm does not know if it is completed. The algorithm needs one whole pass without any swap to know it is sorted.

Third Pass:

(1 2 4 5 8) –> (1 2 4 5 8)
(1 2 4 5 8) –> (1 2 4 5 8)
(1 2 4 5 8) –> (1 2 4 5 8)
(1 2 4 5 8) –> (1 2 4 5 8)

Source Code

```
//  Created by Kaustav Ghosh Dastidar.
#include <stdio.h>
int main()
{
   int a[100],i,n,step,temp;
   printf("Enter the size of the array element: ");
   scanf("%d",&n);
   printf("%d. Enter the array elements: ",i+1);
   scanf("%d",&a[i]);
   for(step=0;step<n-1;++step)
   for(i=0;i<n-step-1;++i)
   {
     if(a[i]>a[i+1])   /* Change > to < in this line, if you want to sort in descending order,
*/
       {
          temp=a[i];
          a[i]=a[i+1];
          a[i+1]=temp;
       }
   }
   printf("In ascending order: ");
   for(i=0;i<n;++i)
       printf("%d  ",a[i]);
   return 0;
}
```

Output

Enter the size of the array element: 5

1. Enter the array elements: 5 1 4 2 8

In ascending order: 1 2 4 5 8

C Program-To Sort Elements Using Insertion Sort

Code Description

Insertion sort is used to sort a list of unsorted numbers and arrange those numbers in ascending or descending order.

It's starts from the ***second element***, compare it with the ***first element*** and ***swap*** it if it is not in order. Similarly, in the next iteration it checks the ***third element*** and place it at the right place in the ***subarray*** of the first and second elements (since the subarray containing the first and second elements is already sorted). This step repeats with the fourth element of the array in the next iteration and place it at the right position in the subarray containing the first, second and the third elements. This process continues until the array gets sorted.

Source code

```
//Created by Kaustav Ghosh Dastidar.

#include <stdio.h>

int main()
{
 int n, array[1000], a, b, t;

 printf("Enter number of elements\n");
 scanf("%d", &n);

 printf("Enter %d numbers to be sorted\n", n);

 for (a = 0; a < n; a++) {

  scanf("%d", &array[a]);
 }

 for (a = 1 ; a <= n - 1; a++) {
  b = a;
```

```
  while ( b > 0 && array[b-1] > array[b]) {
/*To sort elements in descending order change b > 0 && array[b-1] < array[b] in above
line.*/
      t  = array[b];
      array[b]   = array[b-1];
      array[b-1] = t;

      b--;
    }
  }

  printf("Sorted numbers in ascending order:\n");

  for (a = 0; a <= n - 1; a++) {
    printf("%d\n", array[a]);
  }

  return 0;
}
```

Output

```
Enter number of elements
6
Enter 6 numbers to be sorted
89
-5
16
72
25
23
Sorted numbers in ascending order:
-5
16
23
25
72
89
```

C Program-To Sort Elements Using Merge Sort

Code description

This is a very efficient sorting algorithm. All array of n element is split around its center to produce two sub arrays. After these two arrays are sorted independently, they are merged to produce the final sorted list.

Source code

```
//Created by Kaustav Ghosh Dastidar.

#include<stdio.h>
#include<conio.h>
#define max 10
void mergesort(int x[], int n)
{
int aux[max], i, j, k, l1, l2, size, u1, u2;
size = 1; /* Merge array of size 1 */
while (size < n)
   {
        l1 = 0;        /* Initialize lower bounds of first part */
        k = 0;/* k is index for auxiliary array.       */
        while (l1+size < n) {     /* Check to see if there  */
                                  /* are two parts to merge */
 /* Compute remaining indices */
l2 = l1+size;
u1 = l2-1;
u2 = (l2+size-1 < n) ? l2+size-1 : n-1;
/* Proceed through the two subparts */
for (i = l1, j = l2; i <= u1 && j <= u2; k++)
/* Enter smaller-into the array aux */
if (x[i] <= x[j])
aux[k] = x[i++];
else
aux[k] = x[j++];
/* At this point, one of the subparts */
        /* has been exhausted. Insert any     */
```

```
/* remaining portions of the other part */
for (; i <= u1; k++)
        aux[k] = x[i++];
for (; j <= u2; k++)
aux[k] = x[j++];
        /* Advance l1 to the start of the next pair of parts. */
l1 = u2+1;
} /* end while */


/* Copy any remaining value */
for (i = l1; k < n; i++)
aux[k++] = x[i];
/* Copy aux into x and adjust size */
for (i = 0; i < n; i++)
        x[i] = aux[i];
        size *= 2;
        } /* end whi1e */
} /* end mergesort */


void main()
{
        int i,a[max]; ;
        clrscr();
        printf("Enter %d Numbers :\n",max);
        for(i=0;i<max;i++)
        {
        fflush(stdin);
        printf("enter no %d:\t",i+1);
        scanf("%d",&a[i]);
        fflush(stdin);
        }
        mergesort(a,max);
        printf("\nThe Elements in Sorted Form ");
        for(i=0;i<max;i++)
        {
        printf("\n%d",a[i]);
        }
        getch();
      }
```

Output1

```
Enter 10 Numbers :
enter no 1:  10
enter no 2:  9
enter no 3:  8
enter no 4:  7
enter no 5:  6
enter no 6:  5
enter no 7:  3
enter no 8:  1
enter no 9:  4
enter no 10:2

The Elements in Sorted Form
1
2
3
4
5
6
7
8
9
10
```

C Program-To Sort Elements Using Quick Sort

Code description

This is a very efficient sorting technique which is based on ***partitioning of array of data into smaller arrays***.It picks an element as ***pivot*** and partitions the given array around the picked pivot.Thus a large array is divided into two parts:
1.Array which holds smaller than specified value (pivot).
2.Array which holds values greater than the pivot value.
This partitioning continues until the sub-arrays are sorted.

Source code

```
// Created by Kaustav Ghosh Dastidar.
#include <stdio.h>
void quick_sort(int[],int,int);
int partition(int[],int,int);

int main()
{
int a[50],n,i;
printf("Enter the no of elements you want to enter:");
scanf("%d",&n);
printf("\nEnter array elements:");

for(i=0;i<n;i++)
scanf("%d",&a[i]);

quick_sort(a,0,n-1);
printf("\nAfter sorting array elements:");

for(i=0;i<n;i++)
printf("%d ",a[i]);

return 0;
}
```

```
void quick_sort(int a[],int l,int h)
{
int j;
if(l<h)
{
j=partition(a,l,h);
quick_sort(a,l,j-1);
quick_sort(a,j+1,h);
}
}

int partition(int a[],int l,int h)
{
int v,i,j,temp;
v=a[l];
i=l;
j=h+1;

do
{
do
i++;

while(a[i]<v&&i<=h);

do
j--;
while(h<a[j]);
```

```
if(i<j)
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
}while(i<j);

a[l]=a[j];
a[j]=v;

return(j);
}
```

Output

Enter the no of elements you want to enter:9

Enter array elements:8 3 9 10 16 6 1 5 4

After sorting array elements:1 3 4 5 6 8 9 10 16

C Program-To Sort Elements Using Selection Sort

Code description

Elements Using Selection Sort algorithm uses a technique to sort a given array by repeatedly finding the minimum element from the unsorted list and then placing each element in it's correct position. This algorithm maintains two subarrays in a given array.

1) The sorted subarray.
2) Remaining subarray which is unsorted.

In each iteration, the minimum element from the unsorted subarray is picked and moved to the sorted subarray.

Source code

```
// Created by Kaustav Ghosh Dastidar.
#include <stdio.h>
int main()
{
int data[100],a,b,steps,temp;
printf("The number of elements to be sorted: ");
scanf("%d",&b);
for(a=0;a<b;a++)
{
printf("%d. Enter element: ",a+1);
scanf("%d",&data[a]);
}
for(steps=0;steps<b;++steps)
for(a=steps+1;a<b;a++)
{
if(data[steps]>data[a])
/* If you want to sort in descending order, change > to <. */
{
```

```
temp=data[steps];
data[steps]=data[a];
data[a]=temp;
}
}
printf("Sorted list in ascending order: ");
for(a=0;a<b;a++)
printf("%d ",data[a]);
return 0;
}
```

Output

The number of elements to be sorted: 6

1. Enter element: 6
2. Enter element: 71
3. Enter element: 8
4. Enter element: 9
5. Enter element: 30
6. Enter element: 89

Sorted list in ascending order: 6 8 9 30 71 89

C-Program
To Implement Stack Using Array

Code description

This "***C Program***" is written to insert element in ***STACK using Array***. A "***Stack***" is a data structure which is used to store data in a particular order. The basic operations of STACK is ***PUSH() and POP()***. The ***PUSH()*** function is used to insert elements in STACK and ***POP()*** function is used to remove elements from STACK. Here we are all printing the elements which popping out from ***STACK.***

Source code

```
// Created By Kaustav Ghosh Dastidar

#include<stdio.h>

int main()
{
   int st[10];
   int tos=-1,i,r;

   printf("Enter range:  ");
   scanf("%d",&r);
   //pushing data into  a stack
   for(i=0;i<r;i++)
   {
      tos++;
      printf("\nEnter data %d element ",i+1);
      scanf("%d",&st[tos]);
   }

   //displaying the stack
   printf("\nElements of the stack are: ");
   for(i=tos;i>=0;i--)
   {
```

```
 printf("\n%d ",st[i]);
    }

    //popping the stack.
    printf("\nThe popped elements are: ");
    while(tos!=-1)
    {
        printf("\nPopping element: %d",st[tos]);
        tos--;
    }
    return 0;
}
```

Output

```
Enter range:  5
Elements of the stack are:
100
56
11
44
23

The popped elements are:
Popping element: 100
Popping element: 56
Popping element: 11
Popping element: 44
Popping element: 23
```

C-Program
To Implement Stack Using Pointer

Code description

This “C-program” will push() elements in STACK and will also pop() out elements from stack by following LIFO (last-in-first-out) approach. Here we have created 4 user-defined function:

push()
pop()
isFull()
isEmpty()
1. push() – This function will insert the element into stack.

2. pop() – This function will delete the element from stack which was entered last. Because stack follows LIFO approach.

3. isFull() – This function is used to check if the “stack” is already full. If its full then it will show the message “STACK OVERFLOW“.

4. isEmpty() – This function is used to check if the “stack” is empty after deleting elements from STACK. If it is totally empty and you try to delete elements from STACK, then it will display the message “STACK UNDERFLOW“.

Source code

```
//Created By Kaustav Ghosh Dastidar


#include<stdio.h>

#define MAX 10

typedef struct stacktype

{
```

```
 int top;
   int info[MAX];
}stack;
void push(stack *,int);
void pop(stack *);
int isFull(stack);
int isEmpty(stack);
/*WE HAVE TO SEND INSTANCE/REFERENCE OF STACK IN THE main()*/
int main()
{
   stack s;       //Instance of stack is created
   int i,x;

   s.top=-1;      //Initialising the stack
   do
   {
      printf("\nPress [1] for PUSH operation in stack:");
      printf("\nPress [2] for POP operation in stack:");
      printf("\nPress [0] to EXIT");
      printf("\n\n\tEnter your choice: ");
      scanf(" %d",&i);
      switch(i)
      {
         case 1:
            printf("\nEnter an element to push in stack: ");
            scanf(" %d",&x);
            push(&s,x);
            break;

         case 2:
            pop(&s);
            break;

         case 0:
            printf("\nTHE END.....");
            break;
```

```
            default:
                printf("\nINVALID CHOICE");
        }
    }while(i!=0);
    return 0;
}
void push(stack *p,int item)
{
    if(isFull(*p))
    {
        printf("\nSTACK OVERFLOW!.....");
        return;
    }
    p->top++;

    printf("\nThe element pushed is %d", p->info[p->top]=item);
}
int isFull(stack s)
{
    return(s.top==MAX-1);
}
void pop(stack *p)
{
    if(isEmpty(*p))
    {
        printf("\nSTACK UNDERFLOW!.....");
        return;
    }
    printf("\nThe element popped is %d",p->info[p->top--]);
    /*STACK follows LIFO that is why the last element which got entered was pooped
out first from STACK */
}
isEmpty(stack s)
{
    return(s.top==-1);
}
```

Output

Press [1] for PUSH operation in stack:
Press [2] for POP operation in stack:
Press [0] to EXIT

Enter your choice: 1

Enter an element to push in stack: 100

The element pushed is 100
Press [1] for PUSH operation in stack:
Press [2] for POP operation in stack:
Press [0] to EXIT

Enter your choice: 1

Enter an element to push in stack: 200

The element pushed is 200
Press [1] for PUSH operation in stack:
Press [2] for POP operation in stack:
Press [0] to EXIT

Enter your choice: 2

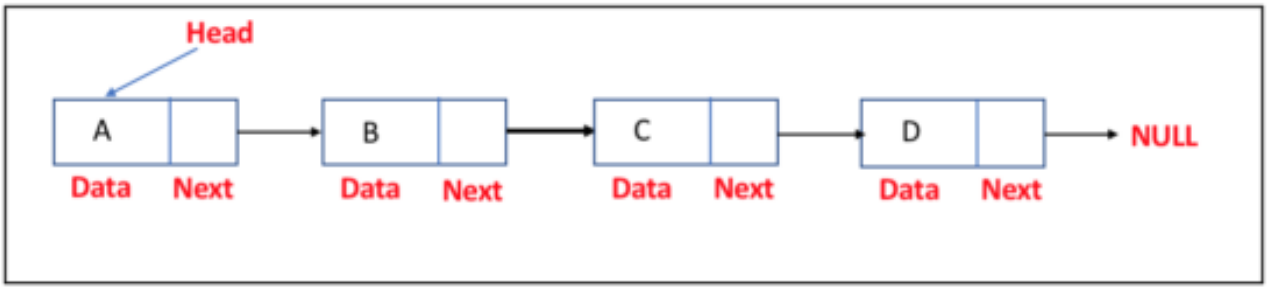
The element popped is 200
STACK follows LIFO that is why the last element which got entered was pooped out first from STACK!
Press [1] for PUSH operation in stack:
Press [2] for POP operation in stack:
Press [0] to EXIT

Enter your choice: 0

C-Program-Linked List Program To Add Modify And Delete Elements In Sigular Linked List

Code description

Linked list is a linear data structure like arrays but they are not stored at contiguous location. All the elements in the "***Linked List***" are linked using pointers by storing the address of next node element.



Significance

Arrays are used to store linear data of similar types but have some limitations:

1. Array is having fixed size and we should know the upper limit on the number of elements entered in array, in advance. The allocated memory size will be equivalent to the upper limit.

2. Also,Inserting a new element in an array is complicated and expensive, because space has to be created for the new element and to create space existing elements have to be shifted.

From example below, in a sorted array of elements num[], it will be difficult to insert new records.

num[] = [100, 200, 300, 400, 500].

Now, if we want to insert a new number 110, then to maintain the sorted order, we have to shift all the elements after 100 (excluding 100).

Deletion is also complex, with arrays unless you are using some other techniques. For example, to delete 200 in num[], everything after 200 has to be shifted.

Advantages over arrays

1. Dynamic size

2. Ease of insertion and deletion or elements

Drawbacks:

1. Accessing elements randomly is not allowed. We have to access elements sequentially starting from the first node. So binary search using linked list is not possible.

2. Extra memory space for a pointer is required with each element of the list.

Source Code

```
//Created By Kaustav Ghosh Dastidar
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#define NULL 0
struct link_list
{
  int item;
  struct link_list *next;
};
typedef struct link_list node;
void entry(node *p);        /*Function prototype declaration*/
void insertfront(node *p);
void insertend(node *p);
void delfront(node *p);
void delend(node *p);
```

```
void insertafter(node *p);
void delafter(node *p);
void print(node *p);
int count(node *p);
void main()
{
 node *head;
 int choice;
 clrscr();
 head=(node*)malloc(sizeof(node));
 entry(head);
 print(head);
 printf("\nThe number of items=%d\n",count(head));
 printf("\n\nEnter [1] to insert a node at the front of the list.");
 printf("\n\nEnter [2] to insert a node at the end of the list.");
 printf("\n\nEnter [3] to delete a node at the front of the list.");
 printf("\n\nEnter [4] to delete a node at the end of the list.");
 printf("\n\nEnter [5] to insert a node at a given position in the list.");
 printf("\n\nEnter [6] to delete a node at a given position in the list.");
 printf("\n\n\t\tPlease enter your choice:");
 scanf("%d",&choice);
 switch(choice)
 {
  case(1):
  insertfront(head);
  break;
  case(2):
  insertend(head);
  break;
  case(3):
  delfront(head);
  break;
  case(4):
  delend(head);
  break;
  case(5):
  insertafter(head);
  break;
  case(6):
  delafter(head);
```

```
}
  //
  getch();
}
void entry(node *p)
{
   printf("\nPlease enter the value.");
   printf("\n(Type -999 if end):");
   scanf("%d",&p->item);
   if(p->item==-999)
   {
      p->next=NULL;
   }
   else
   {
      p->next=(node*)malloc(sizeof(node));
      entry(p->next);
    }
    return;
}
void print(node *p)
{
   if(p->next==NULL)
   printf("End");
   else
   {
    printf("%d------>",p->item);
    print(p->next);
   }
   return;
}
void insertfront(node *p)
{
   node *q;
   clrscr();
   q=(node*)malloc(sizeof(node));
   printf("\nPlease enter the information part of the node:");
   scanf("%d",&q->item);
   q->next=p;
   p=q;
   print(p);
   printf("\n\nThe number of items=%d\n",count(p));
```

```
return;
}
void insertend(node *p)
{
    node *q;
    q=p;
    while(q->next!=NULL)
    {
     q=q->next;
    }
      clrscr();
      printf("\nPlease enter the information part of the node:");
      scanf("%d",&q->item);
      q->next=(node*)malloc(sizeof(node));
      q->next->next=NULL;
      q->next->item=-999;
      print(p);
      printf("\n\nThe number of items=%d\n",count(p));
      return;
}
void delfront(node *p)
{
      node *q;
      q=(node*)malloc(sizeof(node));
      q->item=p->item;
      q->next=p->next;
      p=p->next;
      free(q);
      printf("\n\tThe linked list looks like this after the deletion of the first node\n");
      print(p);
      printf("\nThe number of items=%d\n",count(p));
}
void delend(node *p)
{
      node *q,*s;
      q=p;
      while(q->next->next!=NULL)
      {
          q=q->next;
      }
```

```
 s->next=q->next->next;
            s->item=q->next->item;
            q->item=-999;
            q->next=NULL;
                  free(s);
            printf("\n\n\tThe linked list looks like this after the deletion of the last
node:\n");
            print(p);
            return;
}
void insertafter(node *p)
{
   int i,j,k=0;
   node *q,*s;
   clrscr();
   printf("\nThe linked list looks like below:\n");
   print(p);
   printf("\nPlease enter the number of the node  after which the new node will be \
ninserted:");
   scanf("%d",&i);
   q=p;
   j=i-1;
   while(k<j-1)
   {
     q=q->next;
     k++;
   }
   printf("\nPlease enter the information part of the new node:");
   scanf("%d",&s->item);
   s->next=q->next;
   q->next=s;
   printf("\nThe link list after insertion is given bellow:\n");
   print(p);
   printf("\nThe number of items=%d\n",count(p));
   return;
}
void delafter(node *p)
{
```

```
 int i,k=0,j;
       node *q,*s;
       clrscr();
       q=p;
       printf("\nThe list is given bellow:");
       print(p);
       printf("\nPlease enter the number of the node  which you want to delete:");
       scanf("%d",&i);
       j=i-1;
       while(k<j-1)
       {
          q=q->next;
          k++;
       }
       s->next=q->next->next;
       s->item=q->next->item;
       q->next=q->next->next;
       free(s);
       printf("\nThe List looks like this after deletion:\n");
       print(p);
       return;
}
int count(node *p)
{
    if(p->next==NULL)
    return(0);
    else
    return(1+count(p->next));
}
```

Output 1

The below screenshot is the output of the program of adding data to the “Linked List”. Type “-999” to mark as the end of “Data” addition into the “Linked List”.

```
Please enter the value.
(Type -999 if end):10

Please enter the value.
(Type -999 if end):20

Please enter the value.
(Type -999 if end):-999
10------>20------>End
The number of items=2


Enter [1] to insert a node at the front of the list.

Enter [2] to insert a node at the end of the list.

Enter [3] to delete a node at the front of the list.

Enter [4] to delete a node at the end of the list.

Enter [5] to insert a node at a given position in the list.

Enter [6] to delete a node at a given position in the list.

                Please enter your choice:
```

Output 2

The below screenshot is the ***output*** of the program of adding ***data*** to the front node of the “Linked List”.In the below example, “***900***” value has been added at the front of the linked list.

```
Please enter the information part of the node:900
900------>10------>20------>End

The number of items=3
```

Output 3

The below screenshot is the ***output*** of the program of adding ***data*** to the end node of the “Linked List”.In the below example, “***1000***” value has been added at the end node of the linked list.

```
Please enter the information part of the node:1000
900------>10------>20------>1000------>End

The number of items=4
```

Output 4

The below screenshot is the ***output*** of the program of adding ***data at the given position in the "Linked List***". You have to provide the "***number of the node***" or "***position***" where you have to insert the new node with data. In the below example, we have chosen "***2***" as the number or position where the new node has been inserted (***50***).

```
The linked list looks like below:
900------>10------>20------>1000------>End
Please enter the number of the node  after which the new node will be
inserted:2

Please enter the information part of the new node:50

The link list after insertion is given bellow:
900------>50------>10------>20------>1000------>End
The number of items=5
```

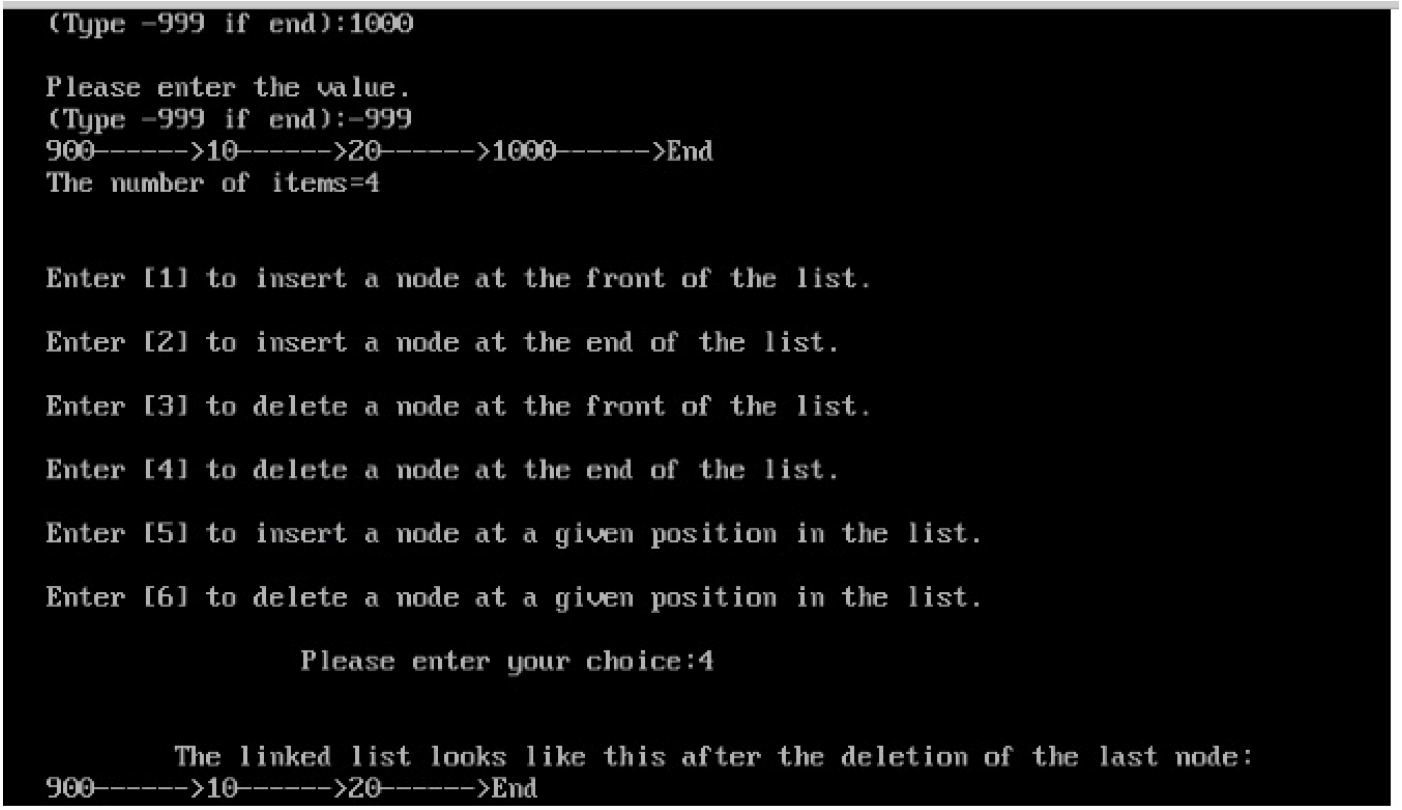
Output 5

The below screenshot is the ***output*** of the program is to ***delete node from the front position of the "Linked List***". In the below example, "***900***" will be deleted from the front.

```
Please enter the value.
(Type -999 if end):-999
900------>10------>20------>1000------>End
The number of items=4


Enter [1] to insert a node at the front of the list.

Enter [2] to insert a node at the end of the list.

Enter [3] to delete a node at the front of the list.

Enter [4] to delete a node at the end of the list.

Enter [5] to insert a node at a given position in the list.

Enter [6] to delete a node at a given position in the list.

                Please enter your choice:3

        The linked list looks like this after the deletion of the first node
10------>20------>1000------>End
The number of items=3
```

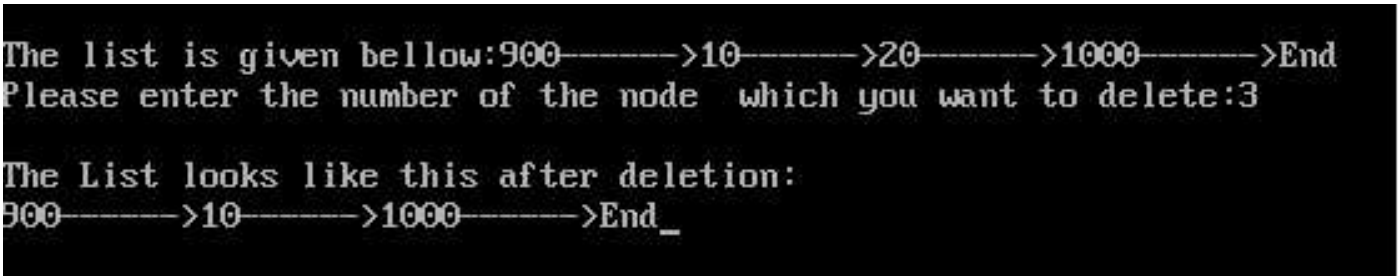
Output 6

The below screenshot is the output of the program is to delete node from the end position of the "Linked List". In the below example, "***1000***" will be deleted from the end.



Output 7

The below screenshot is the output of the program is to delete data ***from the given position in the "Linked List"***. You have to provide the "***number of the node***" or "***position***" where you have to delete the new node with data. In the below example, we have chosen "***3***" as the number or position where the from where the ***node (20)*** can be deleted.



C Program-
Basic File Handling Operations

Code description

In this program we will learn how to add a new record, modify an existing record, delete an existing record and display all record(s) on a file using “C”. “C” is a very powerful language and considered to be the mother of all programming languages since it creates the base of a programmer.

In the below program we have used several functions, lets understand each operation in detail:
fopen(): This function is used for opening a file.

Syntax:FILE pointer_name=fopen (“file_name”,”Mode”);

Example: fp=fopen(“e:/c/kaustav.txt”,”rb+”)

The various kinds of “Opening Modes” in Standard I/O are listed in below table

File	Meaning of Mode	During In existence of file
r	Open for reading.	If the file does not exist, fopen() returns NULL.
rb	Open for reading in binary mode.	If the file does not exist, fopen() returns NULL.
w	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb	Open for writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.

a	Open for append. i.e, Data is added. to end of file.	If the file does not exists, it will be created.
ab	Open for append in binary mode. i.e, Data is added to end of file.	If the file does not exists, it will be created.
r+	Open for both reading and writing.	If the file does not exist, fopen() returns NULL.
rb+	Open for both reading and writing in binary mode.	If the file does not exist, fopen() returns NULL.
w+	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb+	Open for both reading and writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	If the file does not exists, it will be created.
ab+	Open for both reading and appending binary mode.	If the file does not exists, it will be created.

fclose: This function is used for closing an open file

Syntax: fclose(fp)

f(seek): This function will help to get the required data.

Syntax: fseek(FILE * stream, long int offset, int whence)

"FILE* stream" is the pointer to the file.

"**long int offset**" is the position of the record to be found.

"**int whence**" specifies the location where the offset starts.There are 3 different whence in SEEK. The details are given below:

SEEK_SET-Starts the offset from the beginning of the file.

SEEK_END-Starts the offset from the end of the file.

SEEK_CUR-Starts the offset from the current location of the cursor in the file.

Source Code

```
// Created by Kaustav Ghosh Dastidar.

/* Write a menu driven program that will handle an employee file having
   following record structure:
     EMPLOYEE
          [
          int EMP_NO(4),
          char EMP_NAME(30),
          float EMP_SAL(5.2)
          ]
     The operation to be performed on the file are:
          1. Add a new record
          2. Modify an existing record
          3. Delete an existing record
          4. Display all record(s).
*/

#include<stdio.h>

void add_record(FILE *);
void modify_record(FILE *);
void delete_record(FILE *);
void display_record(FILE *);

typedef struct employee
{
int no;
char name[30];
float sal;
}EMP;
```

```
void main()
{
FILE *fp;
int choice;
do
 {
 clrscr();
 fp=fopen("e:/c/kaustav.txt","rb+");
 if(fp==NULL)
   {
   fp=fopen("e:/c/kaustav.txt","wb+");
   if(fp==NULL)
     {
     printf("\n\n Unable to create a file:");
     getch();
     exit(1);
     }
   }
 printf("   ********  MENU  ********\n");
 printf("   ----------------------");
 printf("\n\n Press 1: Add a new record");
 printf("\n\n Press 2: Modify a existing record");
 printf("\n\n Press 3: Delete a existing record");
 printf("\n\n Press 4: Display all record(s)");
 printf("\n\n Press 0: Exit");
 printf("\n\n\t Enter your choice: ");
 scanf("%d",&choice);
 switch(choice)
   {
   case 1: add_record(fp);
           getch();
           break;
   case 2: modify_record(fp);
           getch();
           break;
   case 3: delete_record(fp);
           fclose(fp);
           remove("e:/c/subhendu.txt");
           rename("e:/c/temporary.txt","e:/c/subhendu.txt");
           break;
```

```
 case 4: display_record(fp);
            getch();
            break;
    case 0: printf("\n\n ***** EXIT *****");
            break;
    default:printf("\n\n Invalid choice");
            getch();
    }
 }while(choice);
fclose(fp);
getch();
}
/******************** Add new record ************************/
void add_record(FILE *fp)
{
EMP e;
do
{
clrscr();
fseek(fp,0,SEEK_END);
printf("\n\n Enter the Employee Name: ");
scanf("%s",e.name);
fflush(stdin);
printf("\n\n Enter the Employee No: ");
scanf("%d",&e.no);
fflush(stdin);
printf("\n\n Enter the Employee Salary: ");
scanf("%f",&e.sal);
fflush(stdin);
fwrite(&e,sizeof(e),1,fp);
fseek(fp,0,SEEK_SET);
printf("\n\n Record is added successfully!");
printf("\n\n Do you want to add another record [Y|N]_");
}while(toupper(getchar())!='N');
}
/***************** Dispaly Record ************************/
void display_record(FILE * fp)
{
EMP e;
int count=0;
printf("\n\nEmployee_No        Name       Salary");
printf("\n_____________________________________________"); while(fread(&e,-
sizeof(e),1,fp))
```

```
 {
 printf("\n\n %4d \t %20s \t %5.2f",e.no,e.name,e.sal);
 count++;
 if(!(count%10))
   {
    printf("\n\n press any key to continue");
    getch();
   }
 }
}
/*************** Delete a existing record ***************/
void delete_record(FILE *fp)
{
FILE *tmp;
EMP e;
int empno;
tmp=fopen("e:/c/temporary.txt","wb+");
if(tmp==NULL)
 {
 printf("\n\n Unable to create a temporary file!");
 getch();
 exit(1);
 }
do
 {
  clrscr();
 printf("\n\n Enter searching Employee number: ");
 scanf("%d",&empno);
 fflush(stdin);
 while(fread(&e,sizeof(e),1,fp))
    {
   if(e.no!=empno)
      {
      fwrite(&e,sizeof(e),1,tmp);
      }
    }
 printf("\n\n Searching element is deleted successfully!");
 printf("\n\n Do you want to delete another record [Y|N]_");
 }while(toupper(getchar())!='N');
fclose(tmp);
}
/**************  Modify the Record ******************/
```

```
void modify_record(FILE *fp)
{
int empno;
EMP e,e1;
do
{
printf("\n\n Give the employee number : ");
scanf("%d",&empno);
fflush(stdin);
fseek(fp,0,SEEK_SET);
while(fread(&e,sizeof(e),1,fp))
  {
  printf("\n%d   :",e.no);
  if(e.no==empno)
    {
    e1.no=e.no;
    printf("\n\n Give the new employee name : ");
    scanf("%s",e1.name);
    fflush(stdin);
    printf("\n\n Give the new employee salary :");
    scanf("%d",&e1.sal);
    fflush(stdin);
    fseek(fp,-(sizeof(e1)),SEEK_CUR);
    fwrite(&e1,sizeof(e1),1,fp);
    }
  }
printf("\n\n The desired record is modified!");
printf("\n\n Do you want to continue modification [Y|N]_");
}while(toupper(getchar())!='N');
}
```

Next Steps?

Have any questions while reading the eBook? Want to discuss programming, gadgets and gaming? You can email us (kaustav@gizmofacts.com) where you can ask any questions you have in min2d and subscribe to our Newsletter and visit our Facebook Page.

You can also check our blog "Gizmofacts" where you can find a ton of useful "how-to-guides", tutorials around blogging, programming, gadget reviews and gaming.