```c
// rabin karp pattern matching
// for number at starting
// complexity o(mn)

#include<stdio.h>
#include<string.h>

int hash(char *str,int m,int i)
{
    int j,p=0;
    if(str == NULL){
    printf("error str is null\n");

    }

    for(j=i;j<i+ m;j++)
    p=10*p+(str[j]-'0');

//  printf("%d\n",p);

    return p;

}


int main()
{
    char t[100],p[100]; int m,n,i,pval,tval;
    printf("enter the text");
    scanf("%s",t);
    printf("enter the pattern");
    scanf("%s",p);
    m=strlen(p);
    n=strlen(t);
    pval=hash(p,m,0);
    printf("%d\n",pval);
    tval=hash(t,m,i);
    for(i=0;i<n-m+1;i++)
    {
        if(tval==pval)
        {
        printf("pattern found at index %d",i+1);
        break;
        }
        printf("%d\n",tval);
        tval=10*(tval%(10^m-1))+(t[i+m]-'0');
    }
    if(i==m+n-1)
    printf("pattern doesnot exists");
}
```

```c

//removing duplicates from string bruteforce
//removing adjacent duplicates

#include <stdio.h>
#include<string.h>



int main()
{
    char str[100000];
    int i,j;
    scanf("%s",str);
    j=0;
    for(i=1;i<=strlen(str);i++)
    {
      while(j>=0 && str[i]==str[j])
      {
        i++;j--;
        }
        str[++j]=str[i];

    }


    printf("%s",str);
    return 0;
}
```

```c

//removing duplicates from string bruteforce

#include <stdio.h>

int check(char *str,char c,int i)
{
    int j;
    for(j=0;j<i;j++)
    {
        if(str[j]==c)
        return 1;
    }
    return 0;
}

int main()
{
    char str[100000];
    int i,j,k;
    scanf("%s",str);
    j=1;
    for(i=1;i<strlen(str);i++)
    {

        if(!check(str,str[i],i))
        {
        str[j]=str[i];
        j++;
        }

    }
    str[j]='\0';
    printf("%s",str);
    return 0;
}
```

```c


// finding second largest in array
#include<stdio.h>

int second_largetst_brute_force(int *a,int n)

{
    int prev,i,max;
    if(n==1)
    return -99;
    prev=-99;max=a[0];
    for(i=1;i<n;i++)
    {
        if(a[i]>max)
        {prev=max;
        max=a[i];
        }
    else if(a[i]>prev)
        prev=a[i];
    }
    return prev;

}
int main()
{

    int a[100],n,k,i,small,large;
    printf("enter number of elements(sum elments):");
    scanf("%d",&n);

    small=large=0;
    for(i=0;i<n;i++)
    {
        printf("enter element");
        scanf("%d",&a[i]);
    }

    printf("%d",second_largetst_brute_force(a,n));
    return 0;

}
```

```c
#include<stdio.h>
#include<string.h>
#include<malloc.h>

//note: you can return local array address from a function one way is to use

void reverse(char *str)
{
    int i,j;
    char temp;
    i=0;
    j=strlen(str)-1;

    while(i<j)
    {
        temp=str[i];
        str[i]=str[j];
        str[j]=temp;
        i++;
        j--;
    }
return;
}

char* substr(char *str,int p,int len)
{
    char *temp=malloc((sizeof(char)*len)+1);
    int i,j;
    i=p;
    for(j=0;j<len;j++,i++)
    temp[j]=str[i];
    temp[j]='\0';
    //printf("%s\n",temp);
    return temp;
}

int main()
{
    char str[]="hello";

    char *p=substr(str,1,3);

    printf("%s",p);
}
```

```c
#include<stdio.h>
#include<malloc.h>
#include<stdlib.h>

typedef struct Queue
{
    int front,rear;
    int capacity;
    int *array;
} Queue;

Queue* createQueue(int size)
{
    Queue *Q=(Queue*)malloc(sizeof(Queue));
    if(!Q)
    {
        printf("memeory error!!");
        return NULL;
    }

    Q->front=Q->rear=-1;
    Q->capacity=size;
    Q->array=(int*)malloc(sizeof(int)*Q->capacity);
    if(!Q->array)
    {
        printf("memeory error!!");
        return NULL;
    }
    return Q;
}

int isemptyQueue(Queue *Q)
{
    if(Q)
    {
        if(Q->front==-1)
        return 1;
    }
    return 0;
}

int isfullQueue(Queue *Q)
{
    if(Q)
    {
        if(Q->front==Q->rear+1%Q->capacity)
        return 1;
    }
    return 0;
}
```

```
51:
52: int sizeofQueue(Queue *Q)
53: {
54:         if(Q)
55:         return (Q->capacity-Q->front+Q->rear+1)%Q->capacity;
56:         return 0;
57:
58: }
59:
60: void enqueue(Queue *Q,int item)
61: {
62:     if(isfullQueue(Q))
63:     {printf("queue is full");
64:     return;
65:     }
66:
67:
68:     if(Q->rear==-1)
69:     Q->front=Q->rear=0;
70:     else
71:     Q->rear++;
72:     Q->array[Q->rear]=item;
73:
74:
75: }
76:
77: int dequeue(Queue *Q)
78: {
79:     int x;
80:     if(isemptyQueue(Q))
81:     {
82:         printf("queue is empty");
83:     return -99;
84:     }
85:     x=Q->array[Q->front];
86:     if(Q->front==Q->rear)
87:     Q->front=Q->rear=-1;
88:     else
89:     Q->front++;
90:     return x;
91:
92: }
93:
94: void deleteQueue(Queue *Q)
95: {
96:     if(Q){
97:         if(Q->array);
98:         free(Q->array);
99:     }
100:     free(Q);
```

```
101: }
102:
103:
104: typedef struct tree
105: {
106: int data;
107: struct tree *left,*right;
108: } tree;
109:
110:
111: tree* insert_left(tree *root,int data)
112: {
113:     tree *temp,*newnode;
114:     printf("in left\n");
115:     newnode=(tree*)malloc(sizeof(tree));
116:     if(!newnode){
117:         printf("node could not be created!!");
118:         return;
119:     }
120:    newnode->data=data;
121:     newnode->left=newnode->right=NULL;
122:     if(!root)
123:     {
124:         return newnode;
125:     }
126:     temp=root;
127:      while(root->left)
128:      root=root->left;
129:     root->left=newnode;
130:     return temp;
131: }
132:
133: tree* insert_right(tree *root,int data)
134: {
135:     tree *temp,*newnode;
136:     printf("in right\n");
137:     newnode=(tree*)malloc(sizeof(tree));
138:     if(!newnode){
139:         printf("node could not be created!!");
140:         return;
141:     }
142:    newnode->data=data;
143:     newnode->left=newnode->right=NULL;
144:     if(!root)
145:     {
146:         return newnode;
147:     }
148:     temp=root;
149:      while(root->right)
150:      root=root->right;
```

```
151:     root->right=newnode;
152:     return temp;
153: }
154:
155: void preorder(tree *root)
156: {
157:     if(root)
158:     {
159:         printf("%d-",root->data);
160:         preorder(root->left);
161:         preorder(root->right);
162:     }
163: }
164:
165: int main()
166: {
167:     int i,j,n;
168:     tree *root,*temp;
169:     printf("welcome you are about to create your own tree\n");
170:     printf("enter the root element-");
171:     scanf("%d",&n);
172:     root=(tree*)malloc(sizeof(tree));
173:     if(!root)
174:     {
175:         printf("could not be created!!");
176:         return -1;
177:     }
178:     root->data=n;
179:     root->left=root->right=NULL;
180:     while(1)
181:     {
182:         printf("enter your choice 1.insert a node in left subtree \t2.insert a node
183:         scanf("%d",&i);
184:         switch(i)
185:         {
186:             case 1:
187:                  printf("enter the element to insert:");
188:                  scanf("%d",&n);
189:                  root=insert_left(root,n);
190:                  break;
191:             case 2: printf("enter the element to insert:");
192:                  scanf("%d",&n);
193:                  root=insert_right(root,n);
194:                  break;
195:             case 3:
196:                   preorder(root);
197:                    break;
198:
199:             case 4: exit(0);
200:
```

```
201:             deafault:printf("wrong choice!!");
202:               break;
203:         }
204:
205:     }
206:
207:     return 0;
208:
209: }
```

```c
#include<stdio.h>
#include<malloc.h>



struct node
{
    int data;
    struct node *left;
    struct node *right;
};

struct node* getnode(int data)
{
    struct node *p=(struct node*)malloc(sizeof(struct node));
    if(!p)
    {
        printf("memory error");
        return NULL;
    }
    p->data=data; p->left=p->right=NULL;
    return p;
}

void traverse(struct node *root)
{
    if(root==NULL)
    return;

    traverse(root->left);
    printf("%d-->",root->data);
    traverse(root->right);
}

void mirror(struct node *root)
{
    struct node *temp;
    if(!root)
    return;
    temp=root->left;
    root->left=root->right;
    root->right=temp;
    mirror(root->left);
    mirror(root->right);
}

struct node* clone(struct node *root)
{
    struct node *temp;
    if(!root)
```

```
51:     return NULL;
52:     temp=getnode(root->data);
53:     temp->left=clone(root->left);
54:     temp->right=clone(root->right);
55:     return temp;
56:
57: }
58:
59:
60: int main()
61: {
62: struct node *root=getnode(1),*root1;
63: root->left=getnode(2);
64: root->right=getnode(3);
65: root->left->left=getnode(4);
66: root->left->right=getnode(5);
67: root->right->left=getnode(6);
68: root->right->right=getnode(7);
69: //traverse(root);
70: //mirror(root);
71: traverse(root);
72: printf("\n");
73: root1=clone(root);
74: traverse(root1);
75: return 0;
76: }
```

```
/*
Given a binary tree which has T

nodes, you need to find the diameter of that binary tree. The diameter of a t

Input:
First line contains two integers, T
and X, number of nodes in the tree and value of the root.
Next 2×(T-1)

lines contain details of nodes.
Each detail of node contains two lines. First lines contains a string and sec

String consists of only L
or R. L denotes left child and R

denotes right child. ( Look at the sample explanation for more details )

Output:
Print the diameter of the binary tree.

Constraints:
1=T=20

1=valueofnodes=20

*/









#include <stdio.h>
#include<malloc.h>

struct node
{
    int data;
    struct node *left;
    struct node *right;
};

struct node* getnode(int data)
{
    struct node *p=(struct node*)malloc(sizeof(struct node));
    if(!p)
    {
        printf("memory error");
```

```
51:         return NULL;
52:     }
53:     p->data=data; p->left=p->right=NULL;
54:     return p;
55: }
56:
57: void traverse(struct node *root)
58: {
59:     if(root==NULL)
60:     return;
61:     printf("%d-->",root->data);
62:     traverse(root->left);
63:     traverse(root->right);
64: }
65:
66: int main()
67: {
68:     int T,i,val;
69:     struct node *root,*temp;
70:     char *pos;
71:     scanf("%d%d",&T,&val);
72:     root=getnode(val);
73:     for(i=1;i<T;i++)
74:     {
75:         temp=root;
76:         scanf("%s",pos);
77:         printf("%s\n",pos);
78:         while(*(pos+1)!='\0')
79:         {
80:             if(*pos=='L')
81:             temp=temp->left;
82:             else if(*pos=='R')
83:             temp=temp->right;
84:         }
85:         scanf("%d",&val);
86:         if(*pos=='L')
87:         temp->left=getnode(val);
88:         else if(*pos=='R')
89:         temp->right=getnode(val);
90:
91:     }
92:
93:     traverse(root);
94:     return 0;
95: }
96:
```

```c
#include<stdio.h>
#include<string.h>
#include<malloc.h>

typedef struct trienode
{
    struct trienode *child[26];
    int isleaf;
} node;

int chartoindex(char c)
{
    return ((int)c-(int)'a');
}

node* getNode()
{
    node *newnode=(node*)malloc(sizeof(node));
    int i=0;
    if(!newnode)
    {
        printf("memory error\n");
        return NULL;
    }
    newnode->isleaf=0;
    for(i=0;i<26;i++)
    newnode->child[i]=NULL;

    return newnode;
}

void insert(node *root,char *key)
{
    node *temp;
    int level,length,index;
    length=strlen(key);
    //if(!root)
    //root=getnode();
    temp=root;
    for(level=0;level<length;level++)
    {
        index=chartoindex(key[level]);
        if(temp->child[index]==NULL)
        temp->child[index] =getNode();
        temp=temp->child[index];
    }

    temp->isleaf=1;
}

```

```
51: int search(node *root,char *key)
52: {
53:      node *temp;
54:     int level,length,index;
55:     length=strlen(key);
56:     //if(!root)
57:     //root=getnode();
58:     temp=root;
59:     for(level=0;level<length;level++)
60:     {
61:         index=chartoindex(key[level]);
62:         if(temp->child[index]==NULL)
63:         return 0;
64:         temp=temp->child[index];
65:     }
66:
67:     if(temp!=NULL && temp->isleaf==1)
68:     return 1;
69:     else
70:     return 0;
71: }
72:
73:
74: // Driver
75: int main()
76: {
77:     // Input keys (use only 'a' through 'z' and lower case)
78:     char keys[][8] = {"the", "a", "there", "answer", "any",
79:     "by", "bye", "their","thor"};
80:
81:     char output[][32] = {"Not present in trie", "Present in trie"};
82:
83:
84:     node *root = getNode();
85:
86:     // Construct trie
87:     int i,ARRAY_SIZE=sizeof(keys)/sizeof(keys[0]);
88:     for (i = 0; i < ARRAY_SIZE; i++)
89:        insert(root, keys[i]);
90:
91:     // Search for different keys
92:     printf("%s --- %s\n", "there", output[search(root, "there")] );
93:     printf("%s --- %s\n", "these", output[search(root, "these")] );
94:     printf("%s --- %s\n", "their", output[search(root, "their")] );
95:     printf("%s --- %s\n", "thaw", output[search(root, "thaw")] );
96:
97:     return 0;
98: }
```

```c
//ternary search tree
//pending code
//wrong program
#include<stdio.h>
#include<string.h>
#include<malloc.h>

typedef struct tstnode
{
    char data;
    struct tstnode *left,*eq,*right;
    int isleaf;
} node;


node* getnode(char c)
{
    node *newnode=(node*)malloc(sizeof(node));

    if(!newnode)
    {
        printf("memory error\n");
        return NULL;
    }
    newnode->data=c;
    newnode->eq=newnode->left=newnode->right=NULL;
    newnode->isleaf=0;

    return newnode;
}

node* insert(node *root,char *key)
{
    if(!root)
    {
        root=getnode(*key);
        printf("%c",*key);

    }

    if(root->data>*key)
    {
        root->left=insert(root->left,key);
    }
    else if(root->data<*key)
    root->right=insert(root->right,key);
    else
    {
    if(key+1)
    root->eq=insert(root->eq,key+1);
```

```
51:     else
52:     root->isleaf=1;
53:
54:    }
55:
56: }
57:
58: int search(node *root,char *key)
59: {
60:     if(!root)
61:     return 0;
62:  if(root->data>*key)
63:     {
64:         return search(root->left,key);
65:     }
66:     else if(root->data<*key)
67:     return search(root->right,key);
68:     else
69:     {
70:     if(*(key+1)=='\0')
71:     return root->isleaf;
72:
73:      return search(root->eq,key+1);
74:     }
75:
76: }
77:
78:
79: // Driver
80: int main()
81: {
82:
83:
84:  node *root=NULL;
85:   root=insert(root,"cat");
86:    root=insert(root,"cats");root=insert(root,"up");
87:    root=insert(root,"bug");
88:
89:
90:    search(root,"bug")?printf("found"):printf("not found");
91:
92:     return 0;
93: }
```

```c
//wordbreak.c
//Consider the following dictionary
/*{ i, like, sam, sung, samsung, mobile, ice,
  cream, icecream, man, go, mango}

Input:  ilike
Output: Yes
The string can be segmented as "i like".

Input:  ilikesamsung
Output: Yes
The string can be segmented as "i like samsung" or "i like sam sung".
*/
//do it later

#include<stdio.h>
#include<string.h>
#include<malloc.h>
int containword(char *word)
{
    char dictionary[][32]={"mobile","samsung","sam","sung","man","mango",
                           "icecream","and","go","i","like","ice","cream"};
    int size,i;
    size=sizeof(dictionary)/sizeof(dictionary[0]);
    for(i=0;i<size;i++)
    {
        if(!strcmp(word,dictionary[i]))
          return 1;
    }
    return 0;
}
char* substr(char *str,int p,int len)
{
    char *temp=malloc((sizeof(char)*len)+1);
    int i,j;
    i=p;
    for(j=0;j<len;j++,i++)
    temp[j]=str[i];
    temp[j]='\0';
    //printf("%s\n",temp);
    return temp;
}
int wordbreak(char *word)
{

    int len=strlen(word),i;
    if(len==0)
    return 1;


```

```
51:     for(i=1;i<len;i++)
52:     {
53:
54:         if(containword(substr(word,0,i)) && wordbreak(substr(word,i,len-i)))
55:         return 1;
56:
57:     }
58:
59:     return 0;
60:
61: }
62:
63: int main()
64: {
65:     char *p="ilikesamsung";
66:     wordbreak(p)?printf("yes\n"):printf("no\n");
67: }
```

```c
#include<stdio.h>
int main()
{
    int i,sum=0;
    for(i=100;i<=800;i++)
    {
        if(i%7==0)
        {
            printf("%d--",i);
            sum=sum+i;
        }

    }
    printf("%d",sum);
    return 0;
}
```

```c
//catalan number
//with and witjout dp
#include<stdio.h>

int table[1024];

int catalan(int n)
{
    printf("c(%d)\n",n);
    int i,sum=0;
    if(n==0)
    return 1;

    for(i=1;i<=n;i++)
    sum+= catalan(n-i)*catalan(i-1);

    return sum;

}

int fcatalan(int n)
{
    printf("c(%d)\n",n);
    int i,sum=0;
    if(n==0)
    return 1;

    for(i=1;i<=n;i++)
    {
        if(table[i-1]==0)
        table[i-1]=fcatalan(i-1);

        if(table[n-i]==0)
        table[n-i]=fcatalan(n-i);

    sum+= table[n-i]*table[i-1];
    }

    return sum;

}

int main()
{
    printf("%d",fcatalan(4));
}
```

```c
#include<stdio.h>

int m,n;

int coeff(int n,int k)
{

    printf("%d-%d\n",n,k);


    if (n==k || k==0)
    return 1;

    return coeff(n-1,k-1)+coeff(n-1,k);

}

//using dp
int fcoeff(int i,int k,int table[m][n])
{


printf("%d-%d\n",i,k);

    if (i==k || k==0)
    return 1;



        if(table[i-1][k-1]==-99)
        table[i-1][k-1]=fcoeff(i-1,k-1,table);
        if(table[i-1][k]==-99)
        table[i-1][k]=fcoeff(i-1,k,table);
        table[i][k]=table[i-1][k-1]+table[i-1][k];
        return table[i][k];




}


int main()
{
    int i=5,j=3;
    m=i+1; n=j+1;
    int table[m][n];
    for(i=0;i<m;i++)
    for(j=0;j<n;j++)
    table[i][j]=-99;
```

```
51:     printf("%d-->%d\n",m-1,n-1);
52:     printf("%d",fcoeff(m-1,n-1,table));
53:     return 0;
54: }
```

```c
#include<stdio.h>
#include<math.h>

int gcd(int n,int m)
{
    if(n%m==0)
    return m;
    else gcd(m,n%m);
}

int main()
{
    int n,m;
    printf("enter the numbers to find gcd big number should come first:");
    scanf("%d%d",&n,&m);

    printf("gcd of %d and %d is %d",n,m,gcd(n,m));

}
```

```c
//creating heap and heapsort
#define maxheap 0
#define minheap 1

#include<malloc.h>
#include<stdio.h>

typedef struct heap
{
    int *array;
    int capacity;
    int count;
    int heaptype;
} heap;

void swap(int *x,int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}

heap* createheap(int capacity,int type)
{
   heap *H=(heap*)malloc(sizeof(heap));
    if(!H)
        { printf("memory error");
          return NULL;
        }

   H->array=(int *)malloc(sizeof(int)*capacity);
   if(!H->array)
   { printf("memory error");
          return NULL;
        }

    H->capacity=capacity;
    H->count=0;
    H->heaptype=type;
    return H;

}


int parent(heap *H,int i)
{
    if(i<=0 && H->count<i)
    return -1;
    else
```

```
51:     return (i-1)/2;
52: }
53:
54: int leftchild(heap *H,int i)
55: {
56:     int l=2*i+1;
57:     if(l>=H->count)
58:     return -1;
59:     else
60:     return l;
61: }
62:
63: int rightchild(heap *H,int i)
64: {
65:     int r=2*i+2;
66:     if(r>=H->count)
67:     return -1;
68:     else
69:     return r;
70: }
71:
72: int getmax(heap *H)
73: {
74: if(H->count==0)
75: return -1;
76: return H->array[0];
77: }
78:
79: void percolatedown(heap *H,int i)
80: {
81:     int l,r,max,temp;
82:     //printf("*%d*\n",i);
83:     l=leftchild(H,i);
84:     r=rightchild(H,i);
85:     max=i;
86:     if(l!=-1 && H->array[l]>H->array[max])
87:     max=l;
88:     if(r!=-1  && H->array[r]>H->array[max])
89:     max=r;
90:     if(max!=i)
91:     {
92:         swap(&H->array[i],&H->array[max]);
93:         percolatedown(H,max);
94:     }
95:     else
96:     return;
97:
98: }
99:
100: int deletemax(heap *H)
```

```
101: {
102:     int temp;
103:     if(!H || H->count==0)
104:     return -1;
105:     temp=H->array[0];
106:     swap(&H->array[0],&H->array[H->count-1]);
107:     H->count--;
108:     percolatedown(H,0);
109:     return temp;
110:
111: }
112:
113:
114:
115:
116: void resizeheap(heap *H)
117: {
118:     int *arrayold=H->array,i;
119:     H->array=(int*)malloc(sizeof(int)*H->capacity*2);
120:     if(!H->array)
121:     {
122:     printf("memory error");
123:     return;
124:     }
125:     for(i=0;i<H->count;i++)
126:     H->array[i]=arrayold[i];
127:     H->capacity=H->capacity*2;
128:     free(arrayold);
129: }
130:
131:
132: void insert(heap *H,int data)
133: {
134:     int i,p;
135:     if(H->count==H->capacity)
136:     {
137:         printf("heap is full please wait\n");
138:         printf("resizing heap......\n");
139:         resizeheap(H);
140:     }
141:
142:     i=H->count;
143:
144:
145:     while(i>0 && data>H->array[(i-1)/2])
146:     {
147:         H->array[i]=H->array[(i-1)/2];
148:         i=(i-1)/2;
149:         //  printf("%d**%d\n",i,(i-1)/2);
150:
```

```
151:     }
152:
153:     H->array[i]=data;
154:     H->count++;
155: }
156:
157: void destroyHeap(heap *H)
158: {
159:     if(!H)
160:     return;
161:     if(H->array)
162:     free(H->array);
163:     free(H);
164:     H=NULL;
165: }
166:
167: heap* buildheap(int *a,int n)
168: {
169:     int i;
170:     heap *H=createheap(n,0);
171:     if(!H)
172:     {printf("memory error\n");
173:     return NULL;
174:     }
175:     printf("created HEAP....\n");
176:     for(i=0;i<n;i++)
177:     H->array[i]=a[i];
178:     H->count=n;
179:     //i=(n-1)/2;
180:     for(i=(n-1)/2;i>=0;i--)
181:     percolatedown(H,i);
182:     return H;
183:
184: }
185:
186: void heapsort(int *a,heap *H,int n)
187: {
188:     int i;
189:
190:     for(i=n-1;i>=0;i--)
191:     {
192:         a[i]=deletemax(H);
193:     }
194:
195:
196:
197: }
198:
199: int main()
200: {
```

```
201:     int n,a[100],i,j;
202:     heap *H;
203:     printf("enter the number of elements");
204:     scanf("%d",&n);
205:     for(i=0;i<n;i++)
206:     {
207:         printf("enter element");
208:         scanf("%d",&a[i]);
209:
210:     }
211:     H=buildheap(a,n);
212:
213:     //printf("%d\n",deletemax(H));
214:
215:
216:
217:
218:     heapsort(a,H,n);
219:
220:
221:
222:       printf("after sorting\n");
223:     for(i=0;i<n;i++)
224:     printf("%d\n",a[i]);
225:
226:     destroyHeap(H);
227:     //printf("Hello World!\n");*/
228:     return 0;
229: }
230:
```

```c
//checking is sqaure

#include<stdio.h>

int issqaure(int a)
{
    if(a<=0)
    return 0;
    int i=1;
    while((i*i)<a)
    i++;
    if(a==(i*i))
    return 1;
    else
    return 0;
}


int main()
{
    int a=1;
    if(issqaure(a))
    printf("yes\n");
    else
    printf("no\n");
}
```

```c
// 0/1 knapsackproblem
#include<stdio.h>

int m,l;

int max(int i,int j)
{
    return i>j?i:j;
}

int knapsack(int *wt,int *val,int W,int n)
{
    printf("knap(%d,%d)\n",W,n);
    if(W==0 || n==0)
    return 0;

    if(wt[n-1]>W)
    return knapsack(wt,val,W,n-1);

    else
    return max(val[n-1]+knapsack(wt,val,W-wt[n-1],n-1),knapsack(wt,val,W,n-1));
}

int fknapsack(int *wt,int *val,int W,int n,int table[m][l])
{
    printf("fknap(%d,%d)\n",W,n);
    if(W==0 || n==0)
    return 0;

    if(wt[n-1]>W)
    {
        if(table[W][n-1]==-99)
        table[W][n-1]=fknapsack(wt,val,W,n-1,table);

        return table[W][n-1] ;
    }


    else
    {

        if(table[W][n-1]==-99)
        table[W][n-1]=fknapsack(wt,val,W,n-1,table);
        if(table[W-wt[n-1]][n-1]==-99)
        table[W-wt[n-1]][n-1]=fknapsack(wt,val,W-wt[n-1],n-1,table);

     table[W][n]=max(val[n-1]+table[W-wt[n-1]][n-1],table[W][n-1]);
    return table[W][n];
   }
}
```

```
51:
52: int main()
53: {
54:     int size,val[]={10,40,50,70,100},wt[]={1,3,4,5,6},W=9,i,j;
55:     size=sizeof(val)/sizeof(val[0]);
56:     m=W+1; l=size+1;
57:     int table[m][l];
58:     for(i=0;i<m;i++)
59:     for(j=0;j<l;j++)
60:     table[i][j]=-99;
61:
62:     printf("%d\n",fknapsack(wt,val,W,size,table));
63:     return 0;
64: }
```

```c


//finding longest common subsequence
//using dynamic programming
//o(n2)
#include<stdio.h>
#include<string.h>
 int m,n;

int max(int i,int j)
{
    return (i>j?i:j);
}

int lcs(char *x,char *y,int i,int j,int table[m+1][n+1])
{

 printf("%d--->%d\n",i,j);
   if(i<=0 || j<=0)
   return 0;

else
{
        if(x[i]==y[j])
        {

            if(table[i-1][j-1]==-99)
            table[i-1][j-1]=lcs(x,y,i-1,j-1,table);
            printf("%d\n",table[i-1][j-1]);

        return 1+table[i-1][j-1];

    }

    else
    {
        if(table[i-1][j]==-99)
            table[i-1][j]=lcs(x,y,i-1,j,table);

        if(table[i][j-1]==-99)
            table[i][j-1]=lcs(x,y,i,j-1,table);

    return  max(table[i][j-1],table[i-1][j]);

    }

}
}


```

```
51: int main()
52: {
53:     char x[]="abbabb";
54:     char y[]="baabab";
55:     int answer,i,j;
56:     m=strlen(x);
57:     n=strlen(y);
58:     int table[m+1][n+1];
59:     //printf("%d-%d\n",m,n);
60:     for(i=0;i<=m;i++)
61:     for(j=0;j<=n;j++)
62:     table[i][j]=-99;
63:
64:
65:     answer=lcs(x,y,m,n,table);
66:     printf("%d",answer);
67: }
68:
69:
```

```
 1:
 2:
 3: //finding longest common subsequence
 4: #include<stdio.h>
 5: #include<string.h>
 6:
 7: int max(int i,int j)
 8: {
 9:     return (i>j?i:j);
10: }
11:
12: int lcs(char *x,char *y,int i,int j)
13: {
14:
15:  printf("%d-%d\n",i,j);
16:    if(i<0 || j<0)
17:    return 0;
18:
19: else
20: {
21:         if(x[i]==y[j])
22:     return 1+lcs(x,y,i-1,j-1);
23:
24:     else
25:     return  max(lcs(x,y,i,j-1),lcs(x,y,i-1,j));
26:
27: }
28: }
29:
30:
31: int main()
32: {
33:     char x[]="abbabb";
34:     char y[]="baabab";
35:     int answer;
36:     answer=lcs(x,y,strlen(x)-1,strlen(y)-1);
37:     printf("%d",answer);
38: }
39:
40:
```

```c
#include<stdio.h>
#include<limits.h>

//matrix chain multiplication
int size;

int mcm(int *p,int i,int j)
{
    printf("%d-%d\n",i,j);
    if(i==j)
    return 0;

    int k,count,min=INT_MAX;

    for(k=i;k<j;k++)
    {
        count = mcm(p,i,k)+mcm(p,k+1,j)+p[i-1]*p[k]*p[j];
        if(count<min)
        min= count;
    }

    return min;
}
//using dynamic programming
//o(n3)
int fastmcm(int *p,int i,int j,int table[size][size])
{
    printf("%d-%d\n",i,j);
    if(i==j)
    return 0;
    int k,min=INT_MAX;

    for(k=i;k<j;k++)
    {
        if(table[i][k]==-99)
        table[i][k]=fastmcm(p,i,k,table);
        if(table[k+1][j]==-99)
        table[k+1][j]=fastmcm(p,k+1,j,table);
        table[i][j]=table[i][k]+table[k+1][j]+p[i-1]*p[k]*p[j];

        if(table[i][j]<min)
        min= table[i][j];
    }

    return min;

}

int main()
{
```

```
51:     int p[]={40,20,30,10,30},i,j;
52:     size=sizeof(p)/sizeof(p[0]);
53:     int table[size][size];
54:     for(i=0;i<size;i++)
55:     for(j=0;j<size;j++)
56:     table[i][j]=-99;
57:     printf("%d",fastmcm(p,1,size-1,table));
58:     return 0;
59: }
```

```c
//maximum contigious sum in an array

#include<stdio.h>
int m;

int max(int i,int j)
{
    return i>j?i:j;
}


int maxsum(int *a,int n,int table[m])
{
    //time 2n that is o(n) and space is o(n) dynamic programming
    int i,maxsum=0;
    table[0]=max(a[0],0);

    for(i=1;i<n;i++)
    table[i]=max(table[i-1]+a[i],0);
    for(i=0;i<n;i++)
    if(maxsum<table[i])
    maxsum=table[i];
    return maxsum;
}

int maxsum1(int *a,int n,int table[m])
{
    //time n that is o(n) and space is o(n) dynamic programming in single scan
    int i,maxsum=0;
    table[0]=max(a[0],0);

    for(i=1;i<n;i++)
    {

    table[i]=max(table[i-1]+a[i],0);
    if(maxsum<table[i])
    maxsum=table[i];
    }

    return maxsum;
}

int maxsum2(int *a,int n,int table[m])
{
    //contigious sum not two contigious numbers
    //time n that is o(n) and space is o(n) dynamic programming in single scan
    int i;
    table[0]=a[0];
    table[1]=max(a[0],a[1]);
    for(i=2;i<n;i++)
```

```
51:     {
52:         table[i]=max(a[i]+table[i-2],table[i-1]);
53:     }
54:
55:
56:     return table[n-1];
57: }
58: int maxsum3(int *a,int n,int table[m])
59: {
60:     //contigious sum not three contigious numbers
61:     //time n that is o(n) and space is o(n) dynamic programming in single scan
62:     int i;
63:     table[0]=a[0];
64:     table[1]=max(a[0],a[1]);
65:     for(i=2;i<n;i++)
66:     {
67:         table[i]=max(a[i]+table[i-2],table[i-1]);
68:     }
69:
70:
71:     return table[n-1];
72: }
73: int main()
74: {
75:     int size,a[]={-2,11,-4,13,-5,2},i;
76:     size=sizeof(a)/sizeof(a[0]);
77:     m=size;
78:     int table[m];
79:
80:
81:     printf("%d\n",maxsum2(a,size,table));
82:
83: }
```

```c
#include<stdio.h>

void maximuminslidingwindow(int *a,int *b,int n,int k)
{
    int i,j,max=-99;
    for(i=0;i<=n-k;i++)
    {
        max=a[i];
        for(j=1;j<k;j++)
        {
            if(max<a[i+j])
            max=a[i+j];
        }
        b[i]=max;
    }

}

int main()
{
    int b[100];
    int a[100],n,k,i;
    printf("enter number of elements:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("enter element");
        scanf("%d",&a[i]);
    }
    printf("thank you\n");
    printf("enter size of window");
    scanf("%d",&k);
    maximuminslidingwindow(a,b,n,k);
    for(i=0;i<=n-k;i++)
    printf("%d\t",b[i]);
    return 0;
}
```

```
//Below is C++ implementation of above idea.
// C++ program to count number of paths in a maze
// with obstacles.
#include<bits/stdc++.h>
using namespace std;
#define R 10
#define C 10

// Returns count of possible paths in a maze[R][C]
// from (0,0) to (R-1,C-1)
int countPaths(int maze[][C])
{
    // If the initial cell is blocked, there is no
    // way of moving anywhere
    if (maze[0][0]==-1)
        return 0;

    // Initializing the leftmost column
    for (int i=0; i<R; i++)
    {
        if (maze[i][0] == 0)
            maze[i][0] = 1;

        // If we encounter a blocked cell in leftmost
        // row, there is no way of visiting any cell
        // directly below it.
        else
            break;
    }

    // Similarly initialize the topmost row
    for (int i=1; i<C; i++)
    {
        if (maze[0][i] == 0)
            maze[0][i] = 1;

        // If we encounter a blocked cell in bottommost
        // row, there is no way of visiting any cell
        // directly below it.
        else
            break;
    }

    // The only difference is that if a cell is -1,
    // simply ignore it else recursively compute
    // count value maze[i][j]
    for (int i=1; i<R; i++)
    {
        for (int j=1; j<C; j++)
        {
            // If blockage is found, ignore this cell
            if (maze[i][j] == -1)
                continue;
```

```
            // If we can reach maze[i][j] from maze[i-1][j]
            // then increment count.
            if (maze[i-1][j] > 0)
                maze[i][j] = (maze[i][j] + maze[i-1][j]);

            // If we can reach maze[i][j] from maze[i][j-1]
            // then increment count.
            if (maze[i][j-1] > 0)
                maze[i][j] = (maze[i][j] + maze[i][j-1]);
        }
    }

    // If the final cell is blocked, output 0, otherwise
    // the answer
    return (maze[R-1][C-1] > 0)? maze[R-1][C-1] : 0;
}

// Driver code
int main()
{
    int maze[R][C] =  { {0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, -1, 0, -1, 0, -1, 0, -1, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, -1, 0, -1, 0, -1, 0, -1, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, -1, 0, -1, 0, -1, 0, -1, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, -1, 0, -1, 0, -1, 0, -1, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0}};
    cout << countPaths(maze);
    return 0;
}
```

```c
#include<stdio.h>

#define n 16

//n queen problem


void printsolution(int board[n][n])
{
    int i,j;
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        printf("%d\t",board[i][j]);
        printf("\n");
    }

}


int issafe(int board[n][n],int row,int col)
{
    int i,j;

    for(i=0;i<col;i++)
    if(board[row][i])
    return 0;

    for(i=row,j=col;i>=0 && j>=0;i--,j--)
    if(board[i][j])
       return 0;
   for(i=row,j=col;i<n && j>=0;i++,j--)
    if(board[i][j])
       return 0;


return 1;
}

int solveNQutil(int board[n][n],int col)
{
    int row,i;
    if(col>=n)
    return 1;

     for(i=0;i<n;i++)

     {
        if(issafe(board,i,col)){

```

```
51:         board[i][col]=1;
52:
53:         if(solveNQutil(board,col+1))
54:         return 1;
55:
56:         board[i][col]=0;
57:
58:         }
59:    }
60:      return 0;
61: }
62:
63:
64: void solveNQ()
65: {
66:     int i,j,board[n][n];
67:     for(i=0;i<n;i++)
68:     {
69:         for(j=0;j<n;j++)
70:         board[i][j]=0;
71:     }
72:
73:     if(!solveNQutil(board,0))
74:     printf("solution doesnot exist\n");
75:
76:     printsolution(board);
77: }
78:
79:
80: int main()
81: {
82:
83:     solveNQ();
84:     return 0;
85: }
```

```c
#include<stdio.h>

void charSwap(char* a, char* b){
    char temp = *a;
    *a = *b;
    *b = temp;
}

int main()
{
    char a='0',b='1';
    printf("%c-%c\n",a,b);
    charSwap(&a,&b);
    printf("%c-%c\n",a,b);
    return 0;
}
```

```c
#include<stdio.h>
#include<math.h>

int isprime (int n)
{
    int i=2;
    while(i<=sqrt(n))
    {
        if(n%i==0)
        return 1;
        i++;
    }
}

int main()
{
    int n;
    printf("enter the number to check weather it is prime:");
    scanf("%d",&n);
    if(isprime(n))
    printf("%d is prime",n);
    else
    printf("%d is  not prime",n);
}
```

```c
//rat maze problem my solution
#include<stdio.h>
#define R 6
#define C 6



int ratmaze(int maze[R][C],int row,int col)
{
    int i,j;
    printf("%d-%d\n",row,col);
    if(row==R-1 && col==C-1 && maze[row][col]==1)
    return 1;
    if(maze[row][col]==0)
    return 0;

    if(maze[row][col]==1)
    {
        if(row<=R-1 && col+1<=C-1)
        {

        if(ratmaze(maze,row,col+1))
        {
        printf("move to (%d,%d)\n",row,col+1);
        return 1;
       }


        else
        maze[row][col+1]=0;
       }

       if(row+1<=R-1 && col<=C-1)
        {

        if(ratmaze(maze,row+1,col))
        {
        printf("move to (%d,%d)\n",row+1,col);
        return 1;
        }
        else
        maze[row+1][col]=0;
      }
    }
}

void solvemaze(int maze[R][C],int row,int col)
{
    if(maze[0][0]==0 || maze[row-1][col-1]==0)
    {
```

```
51:         printf("solution doesnot exists");
52:         return;
53:     }
54:
55:     else
56:     {
57:         if(ratmaze(maze,0,0))
58:         return;
59:         else
60:         {
61:         printf("solution doesnot exists");
62:         return;
63:         }
64:
65:     }
66: }
67:
68: int main()
69: {
70:     int maze[R][C]={{1,0,0,0,0,0},{1,1,0,0,0,0},{0,1,1,0,0,0},{0,1,1,0,0,0},{0,0,1,1,
71:     solvemaze(maze,R,C);
72: }
```

```c
#include<stdio.h>
#include<string.h>

int main()
{
    char str[50];
    int i=0,j=-1,len;
    printf("enter the string les than 50 chars please!!\n");
    scanf("%s",str);

    len=strlen(str);
    //printf("%s\n",str);
    while(i<len)
    {
        if(j==-1 || str[j]!=str[i])
        {
            j++;
            str[j]=str[i];
            i++;

        }
        else{
            while(i<len && str[j]==str[i])
            {
                i++;
            }
            j--;
        }
    }
    str[j+1]='\0';
    printf("the answer is -%s",str);
    return 0;
}
```

```c
//checking duplicates using negation method


#include<stdio.h>

int main()
{
    int n,a[100],i,j,count=0;

    printf("enter the number of elements");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("enter element in the range of 0-%d\n",n-1);
        scanf("%d",&a[i]);

    }


    for(i=0;i<n;i++)
    {
        if(a[a[i]]<0)
        {
            printf("duplicates exist %d\n",a[i]);
            break;
        }
        else if(a[i]==0){
            count++;
            if(count>1)
            {printf("duplicates exist\n");
            break;
            }
        }
        a[a[i]]=0-a[a[i]];
    }
   if(i==n)
   printf("no duplicates\n");



    return 0;
}

```

```c
//checking maximum duplicates range of numbers is 0 to n-1
//problem 8


#include<stdio.h>

int main()
{
    int n,a[100],i,j,max=0;

    printf("enter the number of elements");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("enter element in the range of 0-%d\n",n-1);
        scanf("%d",&a[i]);

    }


   for(i=0;i<n;i++)
   a[a[i]]+=n;

   for(i=1;i<n;i++)
   {
    if(a[i]>a[max])
    max=i;
   }

   for(i=0;i<n;i++)
   a[i]/=n;

   printf(" %d is repeated maximum number of times",a[max]);

    return 0;
}

```

```c
//finding first repeated element
//problem 9 may use hashing
//here brute force is implemented


#include<stdio.h>

int main()
{
    int n,a[100],i,j,max=0;

    printf("enter the number of elements");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("enter element in the range of 0-%d\n",n-1);
        scanf("%d",&a[i]);

    }


   for(i=0;i<n;i++)
   {
    for(j=i+1;j<n;j++)
    {
        if(a[i]==a[j])
        {
            printf("first repeated element is %d",a[i]);
            return;
           }
       }
   }

   printf("NO duplicates");

    return 0;
}

```

```c
//finding the missing number using xor operation
//problem 17
//numbers from 0 to n-1 1 number is missing
//no number is repeated


#include<stdio.h>

int main()
{
    int n,a[100],x,i;
    printf("enter the number of elements");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("enter element in the range of 0-%d\n",n-1);
        scanf("%d",&a[i]);

    }

    for(i=0;i<n-1;i++)
    x^=a[i];

    for(i=0;i<n;i++)
    x^=i;

    printf("the missing number is %d",x);




    return 0;
}

```

```c
//finding number occuring odd number of times using xor operation
//problem 18



#include<stdio.h>

int main()
{
    int n,a[100],x,i;
    printf("enter the number of elements");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("enter element\n");
        scanf("%d",&a[i]);

    }

    for(i=0;i<n;i++)
    x^=a[i];



    printf("the  number is %d",x);



    return 0;
}

```

```c
//find two repeating elements in the array, range is 1 to n
//problem 19

//not solved

#include<stdio.h>

int main()
{
    int n,a[100],i,x,y,size,S=0,P=1;

    printf("enter the number of elements");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("enter element \n");
        scanf("%d",&a[i]);
        }

  size=n-2;
  for(i=0;i<n;i++)
  {
    S=S+a[i];
    P=P*a[i];
  }


    return 0;
}

```

```c
// given an array find two numbers whose sum is k
//using sorting
//find
#include<stdio.h>


void insertionsort(int *a,int n)
{
    int i,j,temp;
    for(i=1;i<n;i++)
    {
        j=i;
        temp=a[i];
        while(j>=0 && a[j-1]>temp)
        {

        a[j]=a[j-1];
        j--;
        }
        a[j]=temp;

    }
}

int findsumK(int *a,int n,int k)
{
    int i,j;
    i=0;
    j=n-1;

    while(i<j)
    {
        if(a[i]+a[j]==k)
        {
            printf("%d and %d are the required numbers",a[i],a[j]);
            return 1;
        }
        else if(a[i]+a[j]<k)
        i++;
        else
        j--;
    }

//  printf("no such numbers");

}


int main()
{
```

```
51:     int n,a[1000],i,j,k;
52:     printf("enter the number of elements");
53:     scanf("%d",&n);
54:     for(i=0;i<n;i++)
55:     {
56:         printf("enter element");
57:         scanf("%d",&a[i]);
58:
59:     }
60:     printf("enter the required sum\n");
61:     scanf("%d",&k);
62:
63:     insertionsort(a,n);
64:     findsumK(a,n,k);
65:
66:
67:
68:     //printf("Hello World!\n");
69:     return 0;
70: }
```

```c
//problem 30
//two elements whose sum is closest to zero

#include<stdio.h>
#include<math.h>
#include<limits.h>

void insertionsort(int *a,int n)
{
    int i,j,temp;
    for(i=1;i<n;i++)
    {
        j=i;
        temp=a[i];
        while(j>0 && a[j-1]>temp)
        {

        a[j]=a[j-1];
        j--;
        }
        a[j]=temp;

    }
}

int main()
{
    int n,a[1000],i,j,k,closestpositivesum=INT_MAX,closestnegativesum=INT_MIN,answer;
    printf("enter the number of elements");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("enter element");
        scanf("%d",&a[i]);

    }

    insertionsort(a,n);
    for(i=0;i<n;i++)
    printf("%d--\n",a[i]);
    i=0;j=n-1;
    while(i<j)
    {
        k=a[i]+a[j];
        if(k>0)
        {
            if(k<closestpositivesum)
            closestpositivesum=k;
            j--;
        }
```

```
51:
52:         else if(k<0)
53:         {
54:             if(k>closestnegativesum)
55:             closestnegativesum=k;
56:             i++;
57:         }
58:
59:         else{
60:             printf("closest sum is %d",a[i]+a[j]);
61:             return;
62:         }
63:
64:     }
65:
66:    answer=abs(closestnegativesum)<abs(closestpositivesum)?closestnegativesum:closestp
67:
68:     printf("closest sum is %d",answer);
69:     //printf("Hello World!\n");
70:     return 0;
71: }
```

```c
// given an array find three numbers whose sum is k
//using sorting
//find
#include<stdio.h>


void insertionsort(int *a,int n)
{
    int i,j,temp;
    for(i=1;i<n;i++)
    {
        j=i;
        temp=a[i];
        while(j>0 && a[j-1]>temp)
        {

        a[j]=a[j-1];
        j--;
        }
        a[j]=temp;

    }
}

int findsumK(int *a,int n,int k)
{
    int i,j;
    i=0;
    j=n-1;

    while(i<j)
    {
        if(a[i]+a[j]==k)
        {
            printf("%d and %d and",a[i],a[j]);
            return 1;
        }
        else if(a[i]+a[j]<k)
        i++;
        else
        j--;
    }
    return 0;

//  printf("no such numbers");

}


int main()
```

```
51: {
52:     int n,a[1000],i,j,k,remainingsum;
53:     printf("enter the number of elements");
54:     scanf("%d",&n);
55:     for(i=0;i<n;i++)
56:     {
57:         printf("enter element");
58:         scanf("%d",&a[i]);
59:
60:     }
61:     printf("enter the required sum\n");
62:     scanf("%d",&k);
63:
64:     insertionsort(a,n);
65:     for(i=n-1;i>=2;i--)
66:     {
67:         remainingsum=k-a[i];
68:        if(  findsumK(a,n-1,remainingsum))
69:        {
70:         printf(" %d is the required numbers",a[i]);
71:         return;
72:        }
73:     }
74:
75:
76:
77:     //printf("Hello World!\n");
78:     return 0;
79: }
```

```c


//numberof trailing zeros
#include<stdio.h>

int nooftrailingzero(int n)
{
    int i=0;
    while(n>0)
    {
        n=n/5;
        i=i+n;
    }
    return i;
}

//given no of trailing zeros find the number whose factorial
int revtrailingzeros(int m)
{
int  n=5*m;
 while(m<nooftrailingzero(n))
 {
    n=n-5;
 }

return n;
}

int main()
{
    int n;
    printf("enter number:");
    scanf("%d",&n);
    printf("m=%d    n=%d",n,revtrailingzeros(n));
}
```

```c
//stock strategies

#include<stdio.h>



void shuffle_array(int *a,int l,int r)
{
    int i,c,q,k,temp;
    if(l==r)
    return;
    c=l+(r-l)/2;
    q=1+l+(c-l)/2;

    for(k=1,i=q;i<=c;i++,k++)
    {
        temp=a[i];
        a[i]=a[c+k];
        a[c+k]=temp;
    }
    shuffle_array(a,l,c);
    shuffle_array(a,c+1,r);

}

int main()
{
    int size,a[]={1,2,3,4,5,6,7,8},i;
    size=sizeof(a)/sizeof(a[0]);
    shuffle_array(a,0,size-1);
    for(i=0;i<size;i++)
    printf("%d\t",a[i]);

}
```

```c
#include<stdio.h>
#include<string.h>

int main()
{
    char *p="goods";
    char a[]="good";
    printf("%d-%d-%d\n",sizeof(p),sizeof(*p),strlen(p));
    printf("%d-%d-%d\n",sizeof(a),sizeof(*a),strlen(a));
    printf( "%d",printf("%d",printf("abcdefghbcncj")));
    return 0;
}
```

```c
//stock strategeis


#include<stdio.h>

struct interval
{
    int buy;
    int sell;
};


void stock_strategy(int *price,int n)
{
    int i,count=0;
    struct interval sol[n/2 +1];
    while(i<n)
    {
        while(i<n-1  && price[i+1]<=price[i])
        i++;

        if(i==n-1)
        {
            printf("no profit from buying\n");
            return;
        }
        sol[count].buy=i++;

        while(i<n  && price[i-1]<=price[i])
        i++;
        sol[count].sell=i-1;

        printf("buy on the day %d and sell on the day %d\n",sol[count].buy,sol[count]
        count++;


    }
}

int main()
{
    int size,price[]={100,180,260,310,40,535,695};
    size=sizeof(price)/sizeof(price[0]);
    stock_strategy(price,size);
}
```

```c
#include<stdio.h>
 int main()
 {
    int a[20],s[20],i,j,n;
    printf("enter the no of days-");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("enter the price on day %d-",i+1);
        scanf("%d",&a[i]);

     }

     for(i=0;i<n;i++)
     {
        j=1;
        while(j<=i && a[i]>a[i-j])
        j++;
        s[i]=j;
     }

     printf("the stock span is as follows:");

     for(i=0;i<n;i++)
    {
        printf(" price on day %d is %d and span is %d\n",i+1,a[i],s[i]);


     }



 }
```

```
/*Maga and Alex are good at string manipulation problems. Just now they have

A string is called unique if all characters of string are distinct.

String s1
is called subsequence of string s2 if s1 can be produced from s2 by removing

.

String s1
is stronger than s2 if s1 is lexicographically greater than s2

.

You are given a string. Your task is to find the strongest unique string whic

Input:

first line contains length of string.
second line contains the string.

Output:

Output the strongest unique string which is subsequence of given string.

Constraints:

1=|S|=100000

All letters are lowercase English letters.*/

#include <stdio.h>

void swap(char *x,char *y)
{
    char temp;
    temp=*x;
    *x=*y;
    *y=temp;
}




int main()
{
    char str[100001];
    int index[100001];
    long long int i,max,j,n,temp;
```

```
51:     scanf("%lld",&n);
52:     for(i=0;i<=n;i++)
53:     {
54:     scanf("%c",&str[i]);
55:     index[i]=i;
56:
57:     }
58:
59:
60:     for(i=0;i<n;i++)
61:     {
62:         for(j=0;j<n-i;j++)
63:         {
64:             if(str[j+1]<str[j])
65:             {
66:             swap(&str[j],&str[j+1]);
67:             temp=index[j];
68:             index[j]=index[j+1];
69:             index[j+1]=temp;
70:
71:             }
72:         }
73:     }
74:
75:     //printf("%s\n",str);
76:    i=n;
77:    printf("%c",str[i]);
78:    i--;
79:    while(i>=1)
80:    {
81:        if(index[i+1]<index[i])
82:        printf("%c",str[i]);
83:        i--;
84:
85:    }
86:     //printf("Hello World!\n");
87:     return 0;
88: }
```

```c
#include<stdio.h>

void toh(char l, char m,char r,int n)
{
    if(n==0) return;
    toh(l,r,m,n-1);
    printf("%c-%c\n",l,r);
    toh(m,l,r,n-1);

}

int main()
{
    int n;
    char l='l',m='m',r='r';
    printf("enter the number of towers:");
    scanf("%d",&n);
    toh(l,m,r,n);
    return 0;
}
```

```c
#include<stdio.h>

int ugly(int n)
{
    while(n%2==0)
    n=n/2;
    while(n%3==0)
    n=n/3;
    while(n%5==0)
    n=n/5;

    if(n==1)
    return 1;
    else return 0;
}

int print_n_ugly(int n){
    //slow in speed
    int i=1,count=0;
    while(count<n)
    {
    printf("%d-%d\n",count,i);
        if(ugly(i))
        count++;
        i++;
    }

    return i-1;

}

int fprint_n_ugly(int n)
{
    //using dynamic programming time o(n) space o(n)
    int table[n+1],i=1,j=1,k=1,l;

    for(l=0;l<=n;l++)
    table[l]=0;
    i=j=k=table[1]=1;
    for(l=2;l<=n;l++)
    {
        table[l]=2*table[i]<3*table[j]?(2*table[i]<5*table[k])?2*table[i]:5*table[k]:
        if(table[l]==2*table[i])
        i++;
        else if(table[l]==3*table[j])
        j++;
        else
        k++;
        if(table[l]==table[l-1])
        l=l-1;
```

```
51:         //printf("%d\n",table[l]);
52:         /*if((2*i)<(3*j))
53:         {
54:             if((2*i)<(5*k))
55:             {
56:                 table[l]=2*i;
57:                 i++;
58:             }
59:             else
60:             {
61:                 table[l]=5*k;
62:                  k++;
63:             }
64:         }
65:         else if((3*j)<(5*k))
66:         {
67:             table[l]=3*j;
68:             j++;
69:         }
70:         else
71:         {
72:             table[l]=5*k;
73:             k++;
74:         }*/
75:
76:     }
77:
78: //  for(l=1;l<=n;l++)
79: //  printf("%d-\t",table[l]);
80:     return table[n];
81: }
82:
83: int main()
84: {
85:     int a=150;
86:     printf("%d",fprint_n_ugly(a));
87:     return 0;
88: }
```

```c
#include<stdio.h>

char a[4]={'0','0','0','\0'};

void binary(int n)
{
    if(n<1)

    printf("%s\n",a);

    else{
        a[n-1]='0';
        binary(n-1);
        a[n-1]='1';
        binary(n-1);
    }
}

int main()
{
    binary(4);
    return 0;
}
```

```c
#include<stdio.h>
#include<string.h>

//wrong programm


void combinations(char *combination,char *original,int depth,int start)
{
    int i,length=strlen(original);

    //printf("%d",depth);

    for(i=start;i<length;i++)
    {

    combination[depth]=original[i];
    combination[depth+1]="\0";
    printf("%s\n",combination);
    if(i<length-1)
    combinations(combination,original,depth+1,start+1);
    }

}

int main()
{
    char str[100]="abcd",com[100];
    combinations(com,str,0,0);
}
```

```c
#include<stdio.h>

int main()
{
   int a[100],b[100],c[100],n,k,i;

   printf("enter number of elements");
   scanf("%d",&n);
   printf("enter the range of integers");
   scanf("%d",&k);
   for(i=0;i<n;i++)
   {
    printf("enter element:");
    scanf("%d",&a[i]);
   }

   printf("array before sorting");
   for(i=0;i<n;i++)
   {
    //printf("enter element:");
    printf("%d\n",a[i]);
   }

   for(i=0;i<=k;i++)
   c[i]=0;

   for(i=0;i<n;i++)
   c[a[i]]=c[a[i]]+1;

   for(i=1;i<=k;i++)
   c[i]=c[i-1]+c[i];

   for(i=0;i<n;i++)
   {
    b[c[a[i]]]=a[i];
    c[a[i]]--;
   }

   printf("array after sorting");
   for(i=1;i<=n;i++)
   {
    //printf("enter element:");
    printf("%d\n",b[i]);
   }

    //printf("Hello World!\n");
    return 0;
```

```
51:
52:
53:
54:
55:
56:
57: }
58:
59:
```

```c
#include<stdio.h>

int main()
{
    int n,a[100],i,j;

    printf("enter the number of elements");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("enter element in the range of 0-1\n");
        scanf("%d",&a[i]);

    }

    i=0;j=n-1;
    while(i<j)
    {
        while(i<j && a[i]==0)
        i++;
        while(i<j && a[j]==1)
        j--;
        a[i]=0;
        a[j]=1;
        i++;
        j--;
    }

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

 return 0;
}
```

```c
#include<stdio.h>

void swap(int *x,int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}

int main()
{
    int n,a[100],i,j,mid;

    printf("enter the number of elements");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("enter element in the range of 0-2\n");
        scanf("%d",&a[i]);

    }

    i=0;j=n-1,mid=0;




        while(mid<=j)
        {

        switch(a[mid])
        {
            case 0:swap(&a[i],&a[mid]);
            i++;
            mid++;
            break;

            case 1:mid++;
            break;

            case 2:swap(&a[mid],&a[j]);

            j--;
            break;
        }
    }

    for(i=0;i<n;i++)
```

```
51:     printf("%d",a[i]);
52:
53:  return 0;
54: }
```

```c
//finding k smallest elemnts in the array using partition algorithm


// complexity nlogk
#include<stdio.h>

void swap(int *x,int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}

int partition(int *a,int p,int q)

{
    int i,j,m,pivot;
    if(p==q)
    return p;
    i=p;j=i+1;
    pivot=a[p];
   while(j<=q)
   {
    if(a[j]<pivot)
    {
        i++;
        swap(&a[i],&a[j]);
       }
    j++;
   }
   swap(&a[p],&a[i]);
   return i;

}

void printk(int *a,int p,int q,int k)
{
    int m,i;
    if(p<q )
    {
        m=partition(a,p,q);
        if(m==k)
        {
            printf("output-");
            for(i=0;i<k;i++)
            printf("%d\n",a[i]);
        }
        else if(m<k)
        printk(a,m+1,q,k);
```

```
51:         else printk(a,p,m-1,k);
52:     }
53: }
54: int main()
55: {
56:
57:     int a[100],n,k,i,m;
58:     printf("enter number of elements(sum elments):");
59:     scanf("%d",&n);
60:     printf("enter k =");
61:     scanf("%d",&k);
62:
63:     for(i=0;i<n;i++)
64:     {
65:         printf("enter element");
66:         scanf("%d",&a[i]);
67:     }
68:     printk(a,0,n-1,k);
69:
70:
71:     return 0;
72:
73: }
```

```c
//given two sorted arrays find the median of 2n elemnts in logn time

#include<stdio.h>

 int median(int *a,int n)
 {

if(n%2==0)
return (a[n/2]+a[n/2+1])/2;
else
return a[n/2];
      }

int max(int *x,int *y)
      {
        return *x>*y?*x:*y;
      }

int min(int *x,int *y)
      {
        return *x<*y?*x:*y;
      }
 int getmedian(int *a,int *b,int n)
 {
    int m1,m2;
    if(n<0)
    return -1;
    if(n==1)
    return (a[0]+b[0])/2;

    if(n==2)
    return (max(&a[0],&b[0])+min(&a[1],&b[1]))/2;
    m1=median(a,n);
    m2=median(b,n);

    if(m1==m2)
    return m1;

    else if(m1<m2)
    {
     if(n%2==0)
    return getmedian(a+n/2-1,b,n-n/2+1);
    else
    return getmedian(a+n/2,b,n-n/2);
    }

    else
    {
        if(n%2==0)
    return getmedian(b+n/2-1,a,n-n/2+1);
```

```
51:     else
52:     return getmedian(b+n/2,a,n-n/2);
53:     }
54: }
55:
56: int main()
57: {
58:     int a[100]={1,12,15,26,38},b[100]={2,13,17,30,45};
59:     printf("%d",getmedian(a,b,5));
60:     return 0;
61: }
62:
```

```c


// finding smallest and largest in pairs
#include<stdio.h>
int main()
{

    int a[100],n,k,i,small,large;
    printf("enter number of elements(sum elments):");
    scanf("%d",&n);

    small=large=0;
    for(i=0;i<n;i++)
    {
        printf("enter element");
        scanf("%d",&a[i]);
    }

    for(i=0;i<n;i=i+2)
    {
        if(a[i]<a[i+1])
        {
            if(a[i]<a[small])
            small=i;
            else if(a[i+1]>a[large])
            large=i+1;
        }

        else if(a[i]>a[i+1])
        {
            if(a[i]>a[large])
            large=i;
            else if(a[i+1]<a[small])
            small=i+1;
        }
    }
    printf("smallest=%d largest=%d",a[small],a[large]);
    return 0;

}
```

```c
//finding k nearest neighbours of median of a given sorted array
//linear time complexity
#include<stdio.h>
#include<math.h>

 int median(int *a,int n)
 {

if(n%2==0)
return n/2+1;
else
return n/2;
}


      int main()
 {
    int a[10]={1,12,15,26,27},x,i,j,n=5,k,count;
    x=median(a,5);
    printf(" enter the value of k(must be less than %d\n)",n-1);
    scanf("%d",&k);
    i=x-1;
    j=x+1;
    count=0;
    while(i>=0 && j<=n-1 && count<=k)
    {
        if(abs(a[i]-a[x])<abs(a[j]-a[x]))
        {
            printf("%d\n",a[i]);
            i--;
         }
         else if(abs(a[i]-a[x])>=abs(a[j]-a[x]))
         {
                printf("%d\n",a[j]);
            j++;

         }
         count++;
     }

    return 0;
 }

```

```
/* Dynamic Programming solution to find length of the
   longest common substring */
#include<iostream>
#include<string.h>
using namespace std;

// A utility function to find maximum of two integers
int max(int a, int b)
{   return (a > b)? a : b; }

/* Returns length of longest common substring of X[0..m-1]
   and Y[0..n-1] */
int LCSubStr(char *X, char *Y, int m, int n)
{
    // Create a table to store lengths of longest common suffixes of
    // substrings.   Notethat LCSuff[i][j] contains length of longest
    // common suffix of X[0..i-1] and Y[0..j-1]. The first row and
    // first column entries have no logical meaning, they are used only
    // for simplicity of program
    int LCSuff[m+1][n+1];
    int result = 0;  // To store length of the longest common substring

    /* Following steps build LCSuff[m+1][n+1] in bottom up fashion. */
    for (int i=0; i<=m; i++)
    {
        for (int j=0; j<=n; j++)
        {
            if (i == 0 || j == 0)
                LCSuff[i][j] = 0;

            else if (X[i-1] == Y[j-1])
            {
                LCSuff[i][j] = LCSuff[i-1][j-1] + 1;
                result = max(result, LCSuff[i][j]);
            }
            else LCSuff[i][j] = 0;
        }
    }
      for (int i=0; i<=m; i++)
    {
        for (int j=0; j<=n; j++)
        {
         cout<<LCSuff[i][j]<<"\t";
        }
        //cout<<endl;
   }
    return result;
}

/* Driver program to test above function */
int main()
{
    char X[] = "abcde";
    char Y[] = "abc";
```

```
    int m = strlen(X);
    int n = strlen(Y);

    cout << "Length of Longest Common Substring is "
         << LCSubStr(X, Y, m, n);
    return 0;
}
```

```
 1:
 2:
 3: #include<stdio.h>
 4:
 5:
 6: void merge(int *a,int *t,int left,int mid,int right)
 7: {
 8:     int lstart,lend,rstart,rend,temp_pos,size,i;
 9:     lstart=left;
10:     lend=mid-1;
11:     rstart=mid;
12:     rend=right;
13:     temp_pos=left;
14:     size=right-left+1;
15:     while((lstart<=lend) && (rstart<=rend))
16:     {
17:         if(a[lstart]<=a[rstart])
18:         {
19:         t[temp_pos]=a[lstart];
20:         temp_pos++;
21:         lstart++;
22:         }
23:         else
24:         {
25:             t[temp_pos]=a[rstart];
26:             temp_pos++;
27:              rstart++;
28:         }
29:     }
30:                 while(lstart<=lend){
31:                      t[temp_pos]=a[lstart];
32:                     temp_pos++;
33:                      lstart++;
34:                     }
35:                     while(rstart<=rend)
36:                       {
37:                         t[temp_pos]=a[rstart];
38:                             temp_pos++;
39:                             rstart++;
40:                         }
41:
42:
43:
44:             for(i=0;i<=size;i++)
45:             {
46:                 a[right]=t[right];
47:                 right--;
48:             }
49: }
50:
```

```
51:
52: void mergesort(int *a,int *t,int left,int right)
53: {  int mid;
54:     if(right>left)
55:     {
56:         mid=(left+right)/2;
57:         mergesort(a,t,left,mid);
58:         mergesort(a,t,mid+1,right);
59:         merge(a,t,left,mid+1,right);
60:     }
61: }
62:
63: int main()
64: {
65:     int n,a[100],i,t[100];
66:     printf("enter the number of elements");
67:     scanf("%d",&n);
68:     for(i=0;i<n;i++)
69:     {
70:         printf("enter element");
71:         scanf("%d",&a[i]);
72:
73:     }
74:
75:     mergesort(a,t,0,n-1);
76:     for(i=0;i<n;i++)
77:     {
78:         //printf("enter element");
79:         printf("%d\n",a[i]);
80:
81:     }
82:
83:     //printf("Hello World!\n");
84:     return 0;
85: }
```

```c
#include<stdio.h>
#include<string.h>


void swap(char *i,char *j)
{
    char temp;
    temp=*i;
    *i=*j;
    *j=temp;
}

void permute(char *str,int l,int r)
{
    int i;

    if(l==r)
    {

        printf("%s\n",str);
        return;
    }

    for(i=l;i<=r;i++)
    {
        swap(&str[i],&str[l]);//swap
        permute(str,l+1,r);//permute rest
        swap(&str[i],&str[l]);//backtrack
    }

}

int main()
{
    char str[100]="abcd";
    permute(str,0,3);
}
```

```c
#include<stdio.h>
#include<malloc.h>

typedef struct Queue
{
    int front,rear;
    int capacity;
    int *array;
} Queue;

Queue* createQueue(int size)
{
    Queue *Q=(Queue*)malloc(sizeof(Queue));
    if(!Q)
    {
        printf("memeory error!!");
        return NULL;
    }

    Q->front=Q->rear=-1;
    Q->capacity=size;
    Q->array=(int*)malloc(sizeof(int)*Q->capacity);
    if(!Q->array)
    {
        printf("memeory error!!");
        return NULL;
    }
    return Q;
}

int isemptyQueue(Queue *Q)
{
    if(Q)
    {
        if(Q->front==-1)
        return 1;
    }
    return 0;
}

int isfullQueue(Queue *Q)
{
    if(Q)
    {
        if(Q->front==Q->rear+1%Q->capacity)
        return 1;
    }
    return 0;
}

```

```
51: int sizeofQueue(Queue *Q)
52: {
53:         if(Q)
54:         return (Q->capacity-Q->front+Q->rear+1)%Q->capacity;
55:         return 0;
56:
57: }
58:
59: void enqueue(Queue *Q,int item)
60: {
61:     if(isfullQueue(Q))
62:     {printf("queue is full");
63:     return;
64:     }
65:
66:
67:     if(Q->rear==-1)
68:     Q->front=Q->rear=0;
69:     else
70:     Q->rear++;
71:     Q->array[Q->rear]=item;
72:
73:
74: }
75:
76: int dequeue(Queue *Q)
77: {
78:     int x;
79:     if(isemptyQueue(Q))
80:     {
81:         printf("queue is empty");
82:     return -99;
83:     }
84:     x=Q->array[Q->front];
85:     if(Q->front==Q->rear)
86:     Q->front=Q->rear=-1;
87:     else
88:     Q->front++;
89:     return x;
90:
91: }
92:
93: void deleteQueue(Queue *Q)
94: {
95:     if(Q){
96:         if(Q->array);
97:         free(Q->array);
98:     }
99:     free(Q);
100: }
```

```
101:
102: int main()
103: {
104:     Queue *Q=createQueue(6);
105:     enqueue(Q,5);
106:     enqueue(Q,4);
107:     enqueue(Q,3);
108:     enqueue(Q,1);
109:     printf("%d",dequeue(Q));
110:         printf("%d",dequeue(Q));
111:             printf("%d",dequeue(Q));    printf("%d",dequeue(Q));
112:                 printf("%d",dequeue(Q));
113:     return 0;
114: }
```

```c
#include <stdio.h>

void swap(int *x,int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}

int partition(int *a,int p,int q)
{
    int i,j,pivot;

    i=p;
    pivot=a[p];
    j=i+1;
    while(j<=q)
    {
        if(a[j]<pivot)
        {
            i=i+1;
            swap(&a[i],&a[j]);
        }
        j++;
    }
    swap(&a[p],&a[i]);
    return i;

}
void quicksort(int *a,int p,int q)
{
    int m;
    if(p<=q)
    {

    m=partition(a,p,q);
    quicksort(a,p,m-1);
    quicksort(a,m+1,q);
   }

}

void bubblesort(int *a,int n)
{
    int i,j,flag=1;

    for(i=0;i<n-1  && flag;i++)
    {
        flag=0;
```

```
51:         for(j=0;j<n-i-1;j++)
52:         if(a[j]>a[j+1])
53:         {
54:
55:         swap(&a[j],&a[j+1]);
56:         flag=1;
57:         }
58:     }
59: }
60:
61: void selectionsort(int *a,int n)
62: {
63:     int i,j,min;
64:
65:     for(i=0;i<n-1;i++)
66:     {
67:         min=i;
68:         for(j=i+1;j<n;j++)
69:         {
70:             if(a[j]<a[min])
71:             min=j;
72:         }
73:         swap(&a[i],&a[min]);
74:     }
75: }
76:
77: void insertionsort(int *a,int n)
78: {
79:     int i,j,temp;
80:     for(i=1;i<n;i++)
81:     {
82:         j=i;
83:         temp=a[i];
84:         while(j>0 && a[j-1]>temp)
85:         {
86:
87:         a[j]=a[j-1];
88:         j--;
89:         }
90:         a[j]=temp;
91:
92:     }
93: }
94:
95: void shellsort(int *a,int n)
96: {
97:     int i,j,temp,gap;
98:     for(gap=n/2;gap>0;gap/=2)
99:     {
100:
```

```
            for(i=gap;i<n;i++)
            {
                 j=i;
                temp=a[i];
            while(j>=gap && a[j-gap]>temp)
             {

              a[j]=a[j-gap];
                j-=gap;
             }
             a[j]=temp;

    }
  }
}

void merge(int *a,int *t,int left,int mid,int right)
{
    int lstart,lend,rstart,rend,temp_pos;
    lstart=left;
    lend=mid;
    rstart=mid+1;
    rend=right;
    temp_pos;
    while(lstart<=lend && rstart<=rend)
    {
        if(a[lstart]<a[rstart])
        {t[temp_pos]=a[lstart];
        temp_pos++;
        lstart++;
        }
        else
        {
            t[temp_pos]=a[rstart];
            temp_pos++;
             rstart++;
        }
    }
                while(lstart<=lend){
                     t[temp_pos]=a[lstart];
                    temp_pos++;
                     lstart++;
                    }
                    while(rstart<=rend)
                      {
                        t[temp_pos]=a[rstart];
                            temp_pos++;
                            rstart++;
                        }

```

```
151:             while(right>=left)
152:             {
153:                 a[right]=t[right];
154:                 right--;
155:             }
156: }
157:
158:
159: void mergesort(int *a,int *t,int left,int right)
160: {  int mid;
161:     if(left<right)
162:     {
163:         mid=(left+right)/2;
164:         mergesort(a,t,left,mid);
165:         mergesort(a,t,mid+1,right);
166:         merge(a,t,left,mid,right);
167:     }
168: }
169:
170:
171: int main()
172: {
173:     int n,a[1000],i,t[1000];
174:     printf("enter the number of elements");
175:     scanf("%d",&n);
176:     for(i=0;i<n;i++)
177:     {
178:         printf("enter element");
179:         scanf("%d",&a[i]);
180:
181:     }
182:
183:     mergesort(a,t,0,n-1);
184:     for(i=0;i<n;i++)
185:     {
186:         //printf("enter element");
187:         printf("%d\n",a[i]);
188:
189:     }
190:
191:     //printf("Hello World!\n");
192:     return 0;
193: }
194:
```