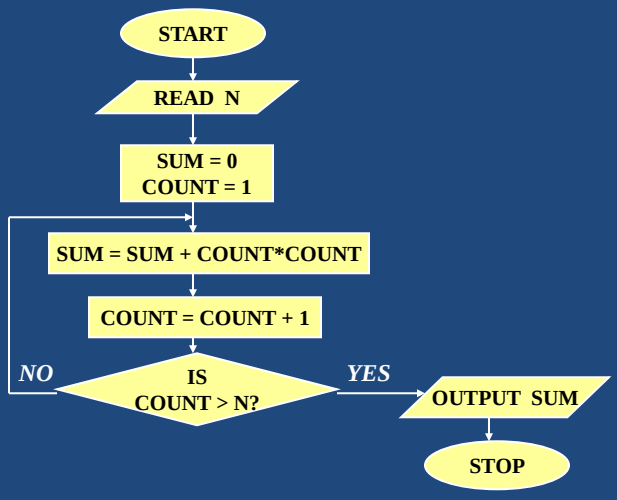
CS11001/CS11002
Programming and Data Structures (PDS) (Theory: 3-0-0)

Class Teacher: Pralay Mitra
Jayanta Mukhopadhyay
Soumya K Ghosh

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

Example 1: $SUM = 1^2 + 2^2 + 3^2 + N^2$

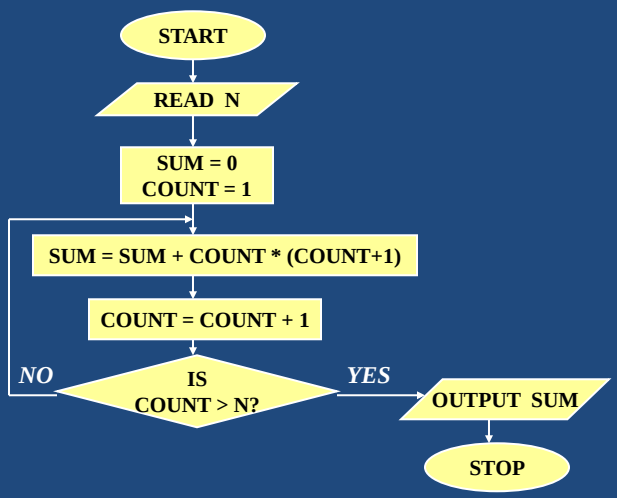


Example 1: *SUM = $1^2 + 2^2 + 3^2 + N^2$*

```
#include <stdio.h>

int main()
{
      int sum, count, N;
      printf("Enter the value of N: ");
      scanf("%d",&N);
      sum=0;
      count=1;
      while(count<=N) {
            sum+=count*count;
            count++;
      }
      printf("Sum is: %d\n",sum);
      return 0;
}
```

Example 2: *SUM = 1*2 + 2*3 + 3*4 + to N terms*

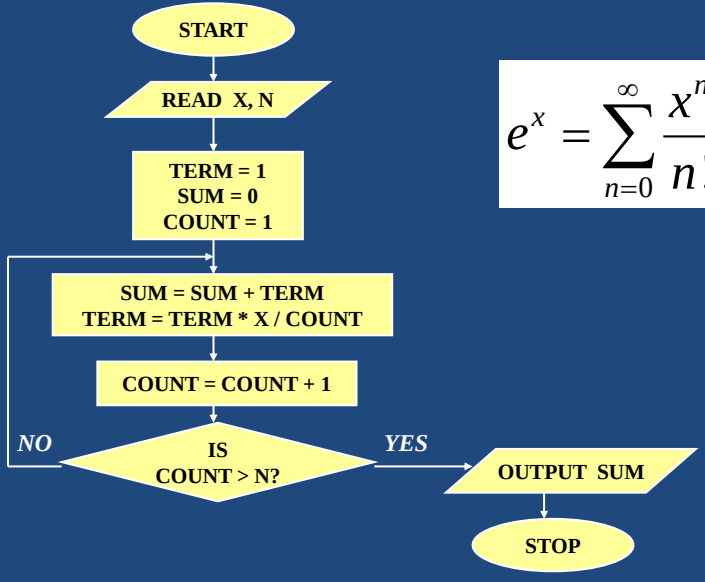


Example 2: *SUM = 1*2 + 2*3 + 3*4 + to N terms*

```
#include <stdio.h>

int main()
{
      int sum, count, N;
      printf("Enter the value of N: ");
      scanf("%d",&N);
      sum=0;
      for(count=1;count<=N;count++) {
            sum+=count*(count+1);
      }
      printf("Sum is: %d\n",sum);
      return 0;
}
```

Example 3: *Computing e^x series up to N terms*



$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

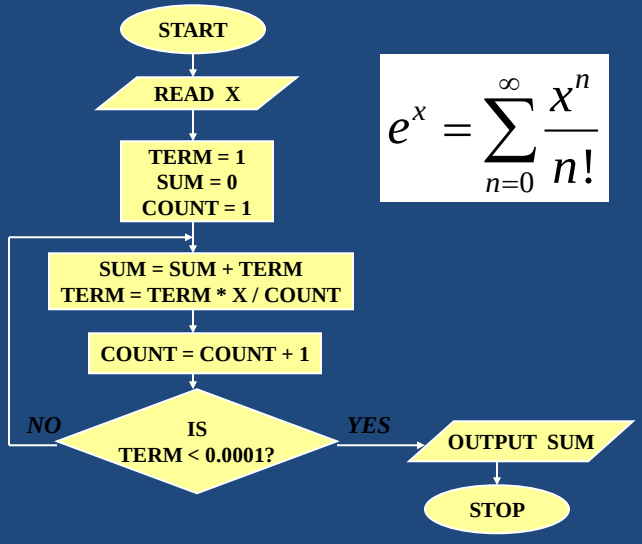
Example 3: *Computing e^x series up to N terms*

```
#include <stdio.h>

int main()
{
      int count, N;
      float x, term, sum;

      printf("Enter the number of terms: ");
      scanf("%d",&N);
      printf("Enter the value of x: ");
      scanf("%f",&x);
      term=1.0;
      sum=0.0;
      count=1;
      while(count<=N) {
            sum+=term;
            term*=x/count;
            count++;
      }
      printf("e^x series upto %d terms is: %10.6f\n",N,sum);
      return 0;
}
```

Example 4: *Computing e^x series up to 4 decimal places*



$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Example 3: *Computing e^x series up to N terms*

```
#include <stdio.h>

int main()
{
      int count;
      float x, term, sum;

      printf("Enter the value of x: ");
      scanf("%f",&x);
      term=1.0;
      sum=0.0;
      count=1;
      while(term>0.0001) {
            sum+=term;
            term*=x/count;
            count++;
      }
      printf("e^x series upto 4 decimal places: %10.6f\n",sum);
      return 0;
}
```

Example 5: computing standard deviation

The Steps

1. Read N
2. Read X_i
3. Compute Mean
4. Compute Standard Deviation

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2}$$

$$\mu = \frac{1}{N}\sum_{i=1}^{N}x_i$$

The Problem

Suppose we have 10 numbers to handle.

Or 20.

Or 100.

How to tackle this problem?

Solution: Use arrays.

Assumption: all the numbers are with same data type.

Arrays

Basic Concept

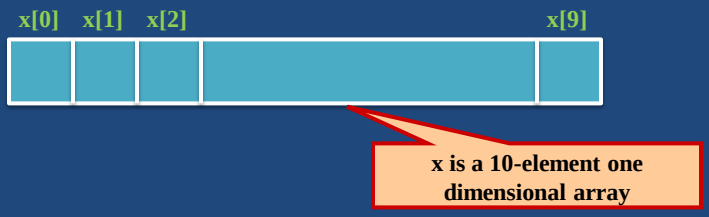
- Many applications require multiple data items that have common characteristics.
  - In mathematics, we often express such groups of data items in indexed form:
    - $x_1, x_2, x_3, \ldots, x_n$
- Why are arrays essential for some applications?
  - Take an example.
  - Finding the minimum of a set of numbers.

Arrays

- Homogenous data types
- All the data items constituting the group share the same name.

```
int x[10];
```

- Individual elements are accessed by specifying the index.



Declaring Arrays

- Like variables, the arrays that are used in a program must be declared before they are used.
- General syntax:
  - type array-name [size];
  - type specifies the type of element that will be contained in the array (int, float, char, etc.)
  - size is an integer constant which indicates the maximum number of elements that can be stored inside the array.

```
int marks[5]; /* marks is an array containing a maximum of 5 integers. */
```

More examples

- Examples:

```
int  x[10];
char  line[80];
float  points[150];
char  name[35];
```

This is not allowed

```
int n;
int   marks[n];
```

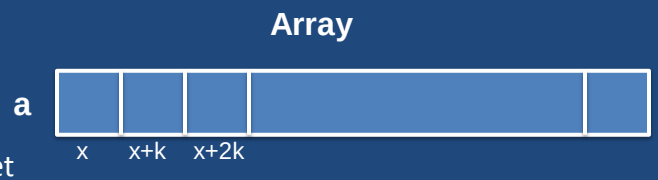
- If we are not sure of the exact size of the array, we can define an array of a large size.

```
int   marks[50];
```

though in a particular run we may only be using, say, 10 elements.

How an array is stored in memory?

- Starting from a given memory location, the successive array elements are allocated space in consecutive memory locations.



- Let
  - x: starting address of the array in memory
  - k: number of bytes allocated per array element
  - Element a[i] :: allocated memory location at address x + i*k
    - First array index assumed to start at zero.

Accessing Array Elements

- A particular element of the array can be accessed by specifying two things:
  - Name of the array.
  - Index (relative position) of the element in the array.
- In C, the index of an array starts from zero.
- Example:
  - An array is defined as `int x[10];`
  - The first element of the array x can be accessed as x[0], fourth element as x[3], tenth element as x[9], etc.
- The array index must evaluate to an integer between 0 and n-1 where n is the number of elements in the array.

```
a[x+2] = 25;
b[3*x-y] = a[10-x] + 5;
```

A Warning

- In C, while accessing array elements, array bounds are not checked.
- Example:

```
int   marks[5];
:
:
marks[8] = 75;
```

  - The above assignment would not necessarily cause an error.
  - Rather, it **MAY** result in unpredictable program results.

Initialization of Arrays

- General form:

  type array_name[size] = { list of values };

- Examples:

  int marks[5] = {72, 83, 65, 80, 76};

  char name[4] = {'A', 'm', 'i', 't'};

- Some special cases:
  - If the number of values in the list is less than the number of elements, the remaining elements are automatically set to zero.

    float total[5] = {24.2, -12.5, 35.1};

    ➔ total[0]=24.2, total[1]=-12.5, total[2]=35.1, total[3]=0, total[4]=0

Initialization of Arrays

- The size may be omitted. In such cases the compiler automatically allocates enough space for all initialized elements.

```
int   flag[] = {1, 1, 1, 0};
char  name[] = {'A', 'm', 'i', 't'};
```

Example 6: Find the minimum of a set of 10 numbers

Array declaration

Reading Array Element

Accessing Array Element

```
#include  <stdio.h>
main()
{
   int  a[10], i, min;
   printf("Give 10 values \n");
   for  (i=0; i<10; i++)
      scanf ("%d", &a[i]);

   min = 99999;
   for  (i=0; i<10; i++)
   {
      if  (a[i] < min)
         min = a[i];
   }
   printf ("\n Minimum is %d", min);
}
```

Alternate Version 1

Change only one line to change the problem size

```
#include  <stdio.h>
#define   SIZE   10

int main()
{
   int  a[SIZE], i, min;
   printf("Give 10 values \n");
   for  (i=0; i<size; i++)
      scanf ("%d", &a[i]);

   min = 99999;
   for  (i=0; i<size; i++)
   {
      if  (a[i] < min)
         min = a[i];
   }
   printf ("\n Minimum is %d", min);
   return 0;
}
```

Alternate Version 2

Define an array of large size and use only the required number of elements

```
#include  <stdio.h>

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

   printf("Give number of elements (n) \n");
   scanf ("%d", &n);  /* Number of elements */
   if(n>100) printf("Array size error!!!");
   printf("Input all n integers \n");
   for  (i=0; i<n; i++)
      scanf ("%d", &a[i]);

   min = 99999;
   for  (i=0; i<n; i++)
   {
      if  (a[i] < min)
         min = a[i];
   }
   printf ("\n Minimum is %d", min);
   return 0;
}
```

Example 7: Computing GPA

Handling two arrays at the same time

```
#include  <stdio.h>
#define  nsub  6

main()
{
   int  grade_pt[nsub], cred[nsub], i,
        gp_sum=0, cred_sum=0, gpa;

printf("Input gr. points and credits for six subjects \n");
   for  (i=0; i<nsub; i++)
      scanf ("%d %d", &grade_pt[i], &cred[i]);

   for  (i=0; i<nsub; i++)
   {
      gp_sum += grade_pt[i] * cred[i];
      cred_sum += cred[i];
   }
   gpa = gp_sum / cred_sum;
   printf ("\n Grade point average:  is %d", gpa);
}
```

Things you cannot do

```
int a[10], b[10];
```

You cannot

- use = to assign one array variable to another

  a = b; /* a and b are arrays */
- use == to directly compare array variables

  if (a = = b) ……..
- directly scanf or printf arrays

  printf (“……”, a);

Array in memory

Accessing Array

```
int a[25],b[25];
```

- How to copy the elements of one array to another?
  - By copying individual elements

```
for (j=0; j<25; j++)
    a[j] = b[j];
```

- By reading them one element at a time

```
for (j=0; j<25; j++)
    scanf (“%f”, &a[j]);
```

  - The ampersand (&) is necessary.
  - The elements can be entered all in one line or in different lines.

Accessing Array

int a[25];

- Printing Array (array elements)
  - By printing them one element at a time.

```
for  (j=0; j<25; j++)
  printf  ("\n %f", a[j]);
```

  - The elements are printed one per line.

```
printf  ("\n");
for  (j=0; j<25; j++)
  printf (" %f", a[j]);
```

  - The elements are printed all in one line (starting with a new line).

Example 5: computing standard deviation

The Steps

1. Read N
2. Read X_i
3. Compute Mean
4. Compute Standard Deviation

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2$$

$$\mu = \frac{1}{N}\sum_{i=1}^{N}x_i$$

Do it now!!!

Exercise Problem

- A shop stores n different types of items. Given the number of items of each type sold during a given month, and the corresponding unit prices, compute the total monthly sales.

STRING

STRINGS

- Array of characters
- The size of array must be predefined.
- Usually one extract character is required to store the null character.
- The null character ('\0') indicates the end of data/string.

STRINGS Library Functions

- Header file is string.h
- Syntax
  - #include <string.h>
- Most frequently used library function:
  - strcmp (to compare between two strings)
  - strcat (to concatenate one string after another)
  - strcpy (to copy one string to another)
  - strlen (determines the length of a string)
  - .....

strcmp

- #include <string.h>
- int strcmp(const char *s1, const char *s2);
  - The strcmp() function compares the two strings s1 and s2. It returns an integer less than, equal to, or greater than zero if s1 is found, respectively, to be less than, to match, or be greater than s2.
- int strncmp(const char *s1, const char *s2, size_t n);

strcat

- #include <string.h>
- char *strcat(char *dest, const char *src);
  - The strcat() function appends the src string to the dest string, overwriting the null byte ('\0') at the end of dest, and then adds a terminating null byte. The strings may not overlap, and the dest string must have enough space for the result.
- char *strncat(char *dest, const char *src, size_t n);

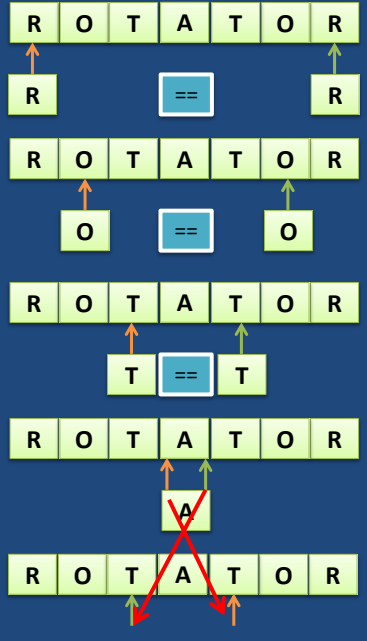
strcpy

- #include <string.h>
- char *strcpy(char *dest, const char *src);

  The strcpy() function copies the string pointed to by src, including the terminating null byte ('\0'), to the buffer pointed to by dest. The strings may not overlap, and the destination string dest must be large enough to receive the copy.
- char *strncpy(char *dest, const char *src, size_t n);

Example 8: Check whether a text is a palindrome or not

- WOW
- MOM
- NOON
- LEVEL
- ANNA
- ROTOR
- ROTATOR



Example 8: Check whether a text is a palindrome or not

```
#include <stdio.h>
#define MAXLEN  100
main()
{
      char text[MAXLEN];
      int i=0,j,len;

      scanf("%s",text);
      while(text[i++]!='\0');                    /* count the length of the text */
      len=i-1;                                   /* length is excluding null character */
      printf("Length of the text is %d\n",len);
      i=0;
      j=len-1;
      while(text[i]==text[j]) {
            i++;
            j--;
            if(i>j)
                  break;
      }
      (i>j)? printf("%s is a palindrome.\n",text) : printf("%s is NOT a palindrome.\n",text);
}
```

Example 9: Palindrome testing for case sensitive cases

- Wow
- Mom
- Noon
- Level
- Anna
- Rotor
- Rotator

1. Check is it a lower case character?
2. If yes convert to upper case.
3. How?
    1. If character is between 65 and 90 (including) then it is in upper case.
    2. If character is between 97 and 122 (including) then it is in lower case.
    3. Subtract 97-65=32
4. Check for palindrome or execute the same code.

Exercise 1: Multiple Word Palindromes

- Was it a cat I saw?
- No lemon, no melon
- Borrow or rob?
- ….

Searching an Array: Linear and Binary Search

Searching

- Check if a given element (key) occurs in the array.

Linear Search

- Basic idea:
  - Start at the beginning of the array.
  - Inspect every element to see if it matches the key.
- Time complexity:
  - A measure of how long an algorithm takes to run.
  - If there are n elements in the array:
    - Best case:
      match found in first element (1 search operation)
    - Worst case:
      no match found, or match found in the last element (n search operations)
    - Average:
      (n + 1) / 2 search operations

Linear Search

```
/* If key appears in a[0..size-1], print its location pos, where a[pos] == key. Else print unsuccessful search */
#include <stdio.h>
int main()
{
    int size,a[100],key,i,pos;

    printf("Enter the number of elements: ");
    scanf("%d",&size);
    printf("Enter the elements: ");
    for(i=0;i<size;i++)
        scanf("%d",&a[i]);
    printf("Enter the key element: ");
    scanf("%d",&key);
    for(pos=-1,i=0;i<size;i++) {   /* initializing pos as unsuccessful search*/
        if(a[i]==key) {
            pos=i;
            break;
        }
    }
    if(pos==-1)
            printf("Unsuccessful search\n");
    else
            printf("The element is present at %d position\n",pos+1);
    return 0;
}
```

Linear Search

```
/* If key appears in a[0..size-1], print its location pos, where a[pos] == key. Else print unsuccessful search */
#include <stdio.h>
#include <stdlib.h>          /* for exit() function */
#define SIZE    100

void main()
{
    int size,a[SIZE],key,i,pos;

    printf("Enter the number of elements: ");
    scanf("%d",&size);
    if(size>SIZE) {       /* size is a variable, SIZE is not!! */
        printf("Array Size error!!! I am exiting .... \n");
        exit(0);
    }
    printf("Enter the elements: ");
    for(i=0;i<size;i++)a
        scanf("%d",&a[i]);
    printf("Enter the key element: ");
    scanf("%d",&key);
    for(pos=-1,i=0;i<size;i++) {   /* initializing pos as unsuccessful search*/
        if(a[i]==key) {
            pos=i;
            break;
        }
    }
    (pos==-1)? printf("Unsuccessful search\n"):printf("The element is present at %d position\n",pos+1);
}
```

1. If function type is void then nothing is to be returned from the function.
2. It is always good to assign a return data type (including void) with each function.
3. exit() will exit from the program without executing remaining statements of the program. This needs stdlib.h as header file.

Linear Search

int x[ ]= {12,-3,78,67,6,50,19,10} ;

- Trace the following calls :

  search 6; → Returns 5

  search 5; → Unsuccessful search

Linear Search

- Basic idea:
  - Start at the beginning of the array.
  - Inspect every element to see if it matches the key.
- Time complexity:
  - A measure of how long an algorithm takes to run.
  - If there are **n** elements in the array:
    - Best case:
      match found in first element (1 search operation)
    - Worst case:
      no match found, or match found in the last element (n search operations)
    - Average:
      (n + 1) / 2 search operations

Search on Sorted List

~~int x[ ]= {12,-3,78,67,6,50,19,10,11} ;~~

int x[ ]= {-3,6,10,~~11~~,12,~~19,50,67,78}~~ ;

- Trace the following calls :

  search 6;

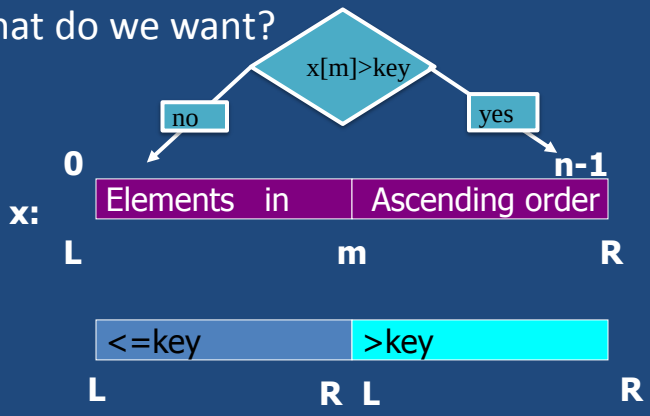
  search 5;

Binary Search

- Binary search works if the array is sorted.
  - Look for the target in the middle.
  - If you don't find it, you can ignore half of the array, and repeat the process with the other half.
- In every step, we reduce the number of elements to search in by half.

The Basic Strategy

- What do we want?



  - Look at [(L+R)/2]. Move L or R to the middle depending on test.
  - Repeat search operation in the reduced interval.

Binary Search

```
/* If key appears in x[0..size-1], prints its location pos where
    x[pos]==key. If not found, print -1 */

int  main ()
{
    int x[100],size,key;
    int L, R, mid;
    _____________________;

    while ( _____________ )
    {
        _____________________;
    }
    _____________________ ;
}
```

The basic search iteration

```
/* If key appears in x[0..size-1], prints its location pos where x[pos]==key. If
     not found, print -1 */

int  main ()
{
     int x[100],size,key;
     int L, R, mid;
     ____________________;
     while ( _____________ )
     {
          mid = (L + R) / 2;
           if  (x[mid] > key)
                R = mid;
           else L = mid;
     }
     ____________________ ;
}
```

Loop termination

```
/* If key appears in x[0..size-1], prints its location pos where x[pos]==key. If
     not found, print -1 */

int  main ()
{
     int x[100],size,key;
     int L, R, mid;
     ____________________;
     while ( L+1 != R )
     {
          mid = (L + R) / 2;
           if  (x[mid] <= key)
                L = mid;
           else R = mid;
     }
     ____________________ ;
}
```

Print result

```
/* If key appears in x[0..size-1], prints its location pos where x[pos]==key. If
     not found, print -1 */

int  main ()
{
     int x[100],size,key;
     int L, R, mid;
     _____;
     while ( L+1 != R )
     {
            mid = (L + R) / 2;
             if  (x[mid] <= key)
                  L = mid;
             else R = mid;
     }
     if   (L >= 0  &&  x[L] = = key)   printf("%d",L);
     else  printf("-1");
}
```

Initialization

```
/* If key appears in x[0..size-1], prints its location pos where x[pos]==key. If not
     found, print -1 */

int  main ()
{
     int x[100],size,key;
     int L, R, mid;
_____;
     L = -1;   R = size;
     while ( L+1 != R )
     {
            mid = (L + R) / 2;
              if  (x[mid] <= key)
                   L = mid;
              else R = mid;
     }
     if   (L >= 0  &&  x[L] = = key)   printf("%d",L);
    else  printf("-1");
}
```

Complete C Program

```
/* If key appears in x[0..size-1], prints its location pos where x[pos]==key. If not found, print -1 */

void main ()
{
      int x[100],size,key;
      int L, R, mid;
      printf("Enter the number of elements: ");
      scanf("%d",&size);
      printf("Enter the elements: ");
      for(i=0;i<size;i++)
            scanf("%d",&a[i]);
      printf("Enter the key element: ");
      scanf("%d",&key);

      L = -1;   R = size;
      while ( L+1 != R )    {
            mid = (L + R) / 2;
               if  (x[mid] <= key)
                     L = mid;
               else R = mid;
      }
      if   (L >= 0  &&  x[L] = = key)   printf("%d",L);
      else  printf("-1");
}
```

Binary Search Examples

Sorted array

-17	-5	3	6	12	21	45	63	50

Trace :

binsearch 3;

binsearch 145;

binsearch 45;

→

L= -1; R= 9; x[4]=12;
L= -1; R=4; x[1]= -5;
L= 1; R=4; x[2]=3;
L=2; R=4; x[3]=6;
L=2; R=3; return L;

Is it worth the trouble ?

- Suppose there are 1000 elements.
- Ordinary search
  - If key is a member of x, it would require 500 comparisons on the average.
- Binary search
  - after 1st compare, left with 500 elements.
  - after 2nd compare, left with 250 elements.
  - After at most 10 steps, you are done.

What is best case?
What is worst case?

Time Complexity

- If there are n elements in the array.
  - Number of searches required:

$\log_2 n$

$2^k = n$,
Where k is the number of steps.

- For n = 64 (say)
  - Initially, list size = 64.
  - After first compare, list size = 32.
  - After second compare, list size = 16.
  - After third compare, list size = 8.
  - …….
  - After sixth compare, list size = 1.

$\log_2 64 = 6$

Homework

Modify the algorithm by checking equality with x[mid].

Download example programs

Copy paste the following link in the web browse:

http://cse.iitkgp.ac.in/~pralay/teaching/2016a/pds/