fsmk.org/labmanual

II
SEMESTER
C PROGRAMMING

LAB MANUAL

VTU SYLLABUS
2010

FREE SOFTWARE MOVEMENT
KARNATAKA

About Free Software Movement Karnataka (FSMK)

Free Software Movement Karnataka (FSMK) is a nonÂprofit organization formed in March 2009 to spread the ideals of free software. We try to increase the understanding of the philosophy behind free software and encourage its use in educational institutions, our very own community center and other organisations.

The phrase â€œfree and open source softwareâ€ can be used to collectively describe a set of operating systems and standalone applications that are free from the clutches of large corporate organisations which produce software only for commercial purposes. Free software is often freelyavailable and is created by a thriving community of programmers, designers and writers. The source code(i.e, the computer program) that underlies an application or an operating system is also open to the perusal of the common individual, which allows every curious tinkerer, student orotherwise, to play around with the source code.Most free software organisations welcome the general public to be part of their community and to contribute in any of the three spheres mentioned above. In the event of you not being a programmer, designer or writer, you can also become a member of the community by actively using free software applications, thereby reporting any issues that you notice, and also by spreading awareness about free software among your friends and family.At FSMK, we believe that it is unfortunate that schools, colleges and other small organisations that can use Linux and related free software for no cost, instead invest in using proprietary and commercial software, thereby spending large amounts annually on licensing and other fees, which can instead be used for the betterment of education or related services, and thereby, the betterment of society.

I want to be involved Website: http://fsmk.org/

Mailing list: http://www.fsmk.org/?q=mailingÂlistÂsubscribe

"About Spoken Tutorial Project:

The Spoken Tutorial project is an initiative of National Mission on Education through ICT, Government of India, to promote IT literacy through Free and Open Source Software. The project is being developed and coordinated by IIT-Bombay and led by Dr. Kannan M. Moudgalya. The project aims at building a repository of self learning courses through video tutorials of various open source softwares. These courses are then used to organize 2 hour workshops in government organizations, NGOs, SMEs and School and Colleges in India completely free of cost for the participants. These tutorials are not only available in English but also in various regional languages for the learner to be able to learn in the language he/she is comfortable in. Currently, Spoken Tutorials are available for free software tools like Blender, GIMP, Latex, Scilab, LibreOffice Suite, Ubuntu Linux, Mozilla Firefox, Thunderbird, MySQL and also programming languages and scripts like C, C++, Python, Ruby, Perl, PHP, Java. All the spoken tutorials which are released under Creative Commons License are available for download free of cost at their website http://spoken-tutorial.org."

GCC- GNU C Compiler

GCC is an alternative to the Turbo C compiler. It provides a variety of features and supports many languages apart from C itself.

When you compile a program , "gcc" checks the source code for errors and creates a binary object file of that code (if no errors exist). It then calls the linker to link your code's object file with other pre-compiled object files residing in libraries. These linked object binaries are saved as your newly compiled program.

Options can be provided to gcc to dictate the way a process is performed. For example, you could tell "gcc" to just create the object file and skip the linking specially when developing large programs or building your own libraries.

Options used in GCC:

The important options commonly used in gcc are-
-*Wall* -L{directory_name}
-*l{library}* -o{file_name}
where:
{library} denotes a library file
{file_name} denotes the name of a Unix file
{directory_name} name of the directory

Example: main.c

```
#include<stdio.h>
int main(void)
    {
       printf("FSMK");
       return 0;
    }
```

Compilation

```
gcc main.c
```

Explanation of the options:

-Wall

> This option enables all the warnings in GCC.

-o

> This is to specify the output file name for the executable.
> ex: **gcc main.c -o main**

-l

> Tells the linker to search a standard list of directories for the library (i.e,used to link with shared libraries). The linker then uses this file as if it had been specified precisely by name.
> ex: **gcc main.c -lccp**

-lm

> To use the library math.h, use the -lm option during compilation.
> ex: **gcc main.c -lm**

-L

> Tells the linker to search standard system directories plus user specified directories.

-g

> Generates additional symbolic debuggging information for use with gdb debugger.

-C

> Produce only the compiled code (without any linking)
> ex: **gcc -C main.c**

-D

> The compiler option -D can be used to define the macro MY_MACRO from command line.
> ex: **gcc -DMY_MACRO main.c**

-fopenmp

> This option is used to enable the different OpenMP directives (#pragma omp). This option along with -static is used to link OpenMP.
> ex: **gcc main.c -fopenmp -o main**

Syntatic differences between Turbo C and GCC:

- The library conio.h is not used in GCC. Hence, getch() and other functions using conio.h do not work.
- The function clrscr() does not work.
- Since GNU/Linux environment always expects a running process to return an exit status when the process is completed, the main() function in C Programs should always return an integer instead of returning void.

Advantages of GCC:

1. GCC is free.
2. Supports multiple languages (C,C++,Java etc)
3. GCC is portable. Runs on almost all platforms.
4. Generates backend code.

Resources

- Please go through the video tutorials on C Programming and GCC developed and released by **Spoken Tutorial Project**, an initiative of National Mission on Education through ICT, Government of India, to promote IT literacy through Open Source Software. Students can go through these video tutorials to get better understanding of the subject. The tutorials can be downloaded from here. More info about the project can be found here

Aim:

Design, develop and execute a program in C to find and output all the roots of a given quadratic equation, for nonzero coefficients.

Summary:

> Any quadratic equation has two roots and the roots of the equation can be found using the formula
>
> ```
> x = (-bÂ±âˆšdiscriminant)/2a
> where discriminant = b2-4ac
> and a,b,c are coefficients when the equation is represented in form
> ax2+bx+c= 0
> ```
>
> Hence as per the equation, if discriminant is equal to 0, then the value of both the roots are equal and real. Also to find x, when discriminant is not 0, we need to find square root of disc. When discriminant value is less than 0, the square root of the discriminant is imaginary and hence the roots of the equation are supposed to be imaginary and distinct. When discriminant value is greater than 0, the square root of the discriminant is real and hence the roots of the equation are supposed to be real and distinct.

Algorithm:

1. Start.
2. Take inputs i.e. coefficient of a, b, and c.
3. When co-efficient a is 0 print error message and go to step 12, else to step 4
4. When coefficient of a is not 0, calculate discriminant
   ```
   desc= b*b-4ac
   ```
5. When desc=0, go to step 6 else go to step 7.
6. The roots are real and equal, roots are
   ```
   x1=x2=-b/2a
   ```
   go to step 11.
7. When desc greater than 0 go to step 8, else to step 9
8. The roots are real and distinct,Roots are
   ```
   x1=(-b+sqrt(desc))/2a,
   x2=(-b-sqrt(desc))/2a
   ```
   go to step 11.
9. When desc lesser than 0 go to step 10.
10. The roots are real and imaginary,roots are
    ```
    x1=p + iq,
    x2= p â€“iq
    ```
    go to step 11.
11. Print roots of given equation.
12. Stop.

Program: roots.c

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

```
#include<stdlib.h>
#include<math.h>

int main()
{
  float a,b,c,desc,x1,x2,r;

  // taking the inputs
  printf("Enter the co-efficient of a,b,c\n");
  scanf("%f%f%f",&a,&b,&c);

  if(a==0)

      // Executes if the equation is linear
      printf("Not a valid quadratic equation\n");
  else
  {
      // Executes for quadratic equation
      desc=(b*b-4*a*c);

      // loop computing equal roots
      if(desc==0)
      {
         // Computation for equal roots
         x1=x2=(-b/2*a);

         // printing equal roots
         printf("Roots are equal and they are\n");
         printf("Root 1= %f and Root 2= %f\n",x1,x2);

      }

      // loop computing distinct roots
      else if(desc>0)
      {

        //Computation for distinct roots
         x1=(-b+sqrt(desc))/(2*a);
        x2=(-b-sqrt(desc))/(2*a);

        // printing distinct roots
        printf("Roots are real and distinct, they are\n");
        printf("Root 1= %f and Root 2= %f\n",x1,x2);

      }

      // loop computing imaginary roots
      else
      {
        // computing real part
        x1=(-b/2*a);
        desc*=-1;

        // computing imaginary part
        r=sqrt(desc)/(2*a);

        // printing imaginary roots
        printf("Roots are imaginary and they are\n");
        printf("Root 1= %f+i%f\n",x1,r);
        printf("Root 2= %f-i%f",x1,r);

      }
```

```
    }
    return 0;
}
```

Output:

Run the following commands in your terminal:

gcc roots.c -lm

./a.out

Enter the coefficients of a,b,c
1 -4 4
Roots are equal and the they are
Root1 = 2.000000 and Root2 = 2.000000

gcc roots.c -lm

./a.out

Enter the coefficients of a,b,c
1 -5 6
The Roots are Real and distinct, they are
Root1 = 3.000000 and Root2 = 2.000000

gcc roots.c -lm

./a.out

Enter the coefficients of a,b,c
1 3 3
The Roots are imaginary and they are
Root1 = -1.500000+i0.866025
Root2 = -1.500000-i0.866025

Aim:

Design, develop and execute a program in C to implement Euclidâ€™s algorithm to find the GCD and LCM of two integers and to output the results along with the given integers.

Summary:

> In its simplest form, Euclid's algorithm starts with a pair of positive integers, and forms a new pair that consists of the smaller number and the difference between the larger and smaller numbers.
> The process repeats until the numbers in the pair are equal. That number then is the greatest common divisor of the original pair of integers.
> The main principle is that the GCD does not change if the smaller number is subtracted from the larger number. For example, the GCD of 252 and 105 is exactly the GCD of 147 (= 252 âˆ’ 105) and 105. Since the larger of the two numbers is reduced, repeating this process gives successively smaller numbers, so this repetition will necessarily stop sooner or later â€” when the numbers are equal (if the process is attempted once more, one of the numbers will become 0).
> Euclidâ€™s Algorithm can thus be simplified by finding the remainder of the two integers and forms a new pair consisting of the divisor and the remainder.
> Also to calculate the LCM of any two numbers m and n, following formula can be used if GCD of the two numbers, GCD(m,n) is already known.
>
> ```
> LCM(m,n) = (m*n)/GCD(m,n)
> ```
>
> To illustrate the extension of Euclid's algorithm, consider the computation of gcd(120, 23), which is shown in the table below. Notice that the quotient in each division is recorded as well alongside the remainder. In this example, the divisor in the last line (which is equal to 1) indicates that the gcd is 1; that is, 120 and 23 are coprime (also called relatively prime).
>
> ```
> Iteration  Dividend  Divisor Remainder  Quotient
>
>    1         120       23       5         5
>
>    2         23        5        3         4
>
>    3         5         3        2         1
>
>    4         3         2        1         1
>
>    5         2         1        0         2
>
>    6         1         0
> ```

Algorithm:

1. Start.
2. Take two numbers as input a and b.
3. If both input numbers are 0 then print error message, go to step 9, else to step 4.
4. Take x=a and y=b.
5. When y is not 0

```
    do
      rem=x%y;
```

```
      x=y;
      y=rem;
```

6. The final value of X is GCD.
7. LCM is computed by

```
   (a*b)/GCD.
```

8. Print the values of GCD and LCM.
9. Stop.

Program: Gcdlcm.c

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

int main()
{
    int x,y,a,b,rem,lcm,gcd;

    // taking inputs
    printf("Enter the two numbers\n");
    scanf("%d%d",&a,&b);

    // statement verifying values of a and b for zero
      if(a==0 && b==0)
    {
        printf("GCD doesn't Exist\n");
    }
    else
    {
        // assigning value of a to x
        x=a;

        // assigning value of b to y
        y=b;

        // loop computing GCD using Euclids algorithm
         while(y!=0)
        {
            rem=x%y;
            x=y;
            y=rem;
        }

        // final value of x when y=0 is GCD
        gcd=x;

        // computing LCM
         lcm=(a*b)/gcd;

         printf("The GCD and LCM of %d and %d is %d and %d\n",a,b,gcd,lcm);

       // prining GCD and LCM of given integer
    }

    return 0;
}
```

Output:

Run the following commands in your terminal:

gcc Gcdlcm.c

./a.out

```
 1.Enter the two numbers
   0 0
   GCD doesn't exist
```

gcc Gcdlcm.c

./a.out

```
2: Enter the two numbers
   64   48
   GCD and LCM of 64 and 48 is 16 and 192
```

gcc Gcdlcm.c

./a.out

```
3: Enter the two numbers
   5   0
   GCD and LCM of 5 and 0 is 5 and 0
```

Aim:

Design, develop and execute a program in C to reverse a given four digit integer number and check whether it is a palindrome or not. Output the given number with suitable message.

Summary:

> Palindrome is a number, word, phrase, or sequence that reads the same backward as forward, e.g., â€œmadamâ€ or â€œmalayalamâ€ or 1221 or 10301.

Algorithm:

1. Start.
2. Take a four digit number as input i.e. n.
3. Assign value of n to m (to retain the value of n)
4. Until the value of n is not 0.

   ```
   Do
   rem=n%10;
   rev=rev*10+rem;
   n=n/10;
   ```

5. When value of m is equal to rev, go to step 6, else to 8.
6. Print given number as palindrome, goto step 8
7. Print given number as not palindrome.
8. Stop.

Program: Palindrome.c

```
#include<stdio.h>
void main()
{
    int n,m,rem,rev=0;
    printf("Enter the number\n");
    scanf("%d",&n); // taking input //
    m=n;   // assigning value of n to m //
    while(n!=0)
    {                   //reversing the integer //
        rem=n%10;
        rev=rev*10+rem;
        n=n/10;
    }

    printf("The reverse of %d is %d\n",m,rev);
           // printing the reversed integer //

    if(m==rev)  //executes when reversed integer is same as given integer //
        printf("It is a palindrome\n");
    else   // executes when not equal //
        printf("It is not a palindrome\n");
}
```

Output:

Run the following commands in your terminal:

gcc palindrome.c

./a.out

```
1. Enter the number
   7667
   Reverse of 7667 is 7667
   It is a palindrome
```

gcc palindrome.c

./a.out

```
2.  Enter the number
    1234
    Reverse of 1234 is 4321
    It is not a palindrome
```

Aim:

Design, develop and execute a program in C to evaluate the given polynomial f(x) = a4x4 + a3x3 + a2x2+ a1x + a0 for given value of x and the coefficients using Hornerâ€™s method.

Summary:

> Horner's method of polynomial evaluation is described below
> The expression is broken down in the following way
> f(x) = a4x4 + a3x3 + a2x2 + a1x + a0
> f(x) = x (a4x3 + a3x2 + a2x + a1) + a0
> f(x) = x (x(a4x2 + a3x2 + a2) + a1) + a0
> f(x) = x (x(x (a4x + a3) + a2) + a1) + a0
> f(x) = x (x(x (x (a4) + a3) + a2) + a1) + a0

Example:

> Input values:
> x=6
> a0=5, a1=4, a2=3, a3=2, a4=1
> sum = a4 *x â†’ 1* 6 â†’ 6
> sum = (sum + a3) *x â†’ (6 + 2)* 6 â†’ 48
> sum = (sum + a2) *x â†’ (48 + 3)* 6 â†’ 306
> sum = (sum + a1) *x â†’ (306 + 4)* 6 â†’ 1860

Algorithm:

1. Start
2. Input the degree and value of x
3. Input Coefficients , upper limit is provided by the value of degree
4. i= degree, sum =0
5. sum = (sum + ai) * x
6. decrement i by 1
7. if (i> 0) goto step 5
8. add a0 to the final sum
9. display sum
10. Stop

Program: polynomial.c

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

int main(void)
{
    int deg,i,Arr[10];
    float x,Sum=0;
    printf("\nEnter the degree of the polynomial and value of x\n");
    scanf("%d%f",&deg,&x); //taking input value of degree and x //
    printf("\nEnter the coefficients in descending order of degree\n");
    for(i=0;i<=deg;i++)
    {
        scanf("%d",&Arr[i]);
    }    // taking co-efficient value polynomial //

    for(i=deg;i>0;i--)
    {
        Sum=(Sum + Arr[i])*x;
    }   //evaluating polynomial using Hornerâ€™s method//
    Sum = Sum + Arr[0]; //adding sum to higher degree co-efficient //
    printf("\nValue of polynomial after evaluation=%g\n",Sum);
    //printing the result//
    return 0;
}
```

Output:

Run the following commands in your terminal:

gcc â€“lm polynomial.c

./a.out

```
1:    Enter the degree of the polynomial and value of x
    5
    2
    Enter the coefficients in descending order of degree
    6 5 4 3 2 1
    Value of polynomial after evaluation=120.000000
```

gcc â€“lm polynomial.c

./a.out

```
2:     Enter the degree of the polynomial and value of x
    4 1
    Enter the coefficients in descending order of degree
    1 2 3 4 5
    Value of polynomial after evaluation=15.000000
```

Aim:

Design, develop and execute a program in C to copy its input to its output, replacing each string of one or more blanks by a single blank.

Summary:

> The program introduces concept of character comparision and also white space characters like â€˜\tâ€™ and null characters like â€˜\0â€™. To find and replace all multiple blanks, with a single blank in the string, we will start from second character as current character in the input string and compare it with the previous character and if only both are not spaces, then the current character will stored in the destination string. We will continue this till we reach the end of the input string which is denoted by null character, â€˜\0â€™.

Algorithm:

1. Start
2. Initialise inspace to 0
3. Take the input string (using gets() function).
4. from i=0 to string length, go to step 5 else go to step 7
5. When position of i is space, else step 6

   Check if inspace is 0

   Assign inspace=1 and print a space, go to step 4

6. Print the character, go to step 4

7. Stop

Program: space.c

```
#include<stdio.h>
#include<string.h>
int main()
{
      char c[50];
      int i,inspace=0;
      printf("Enter the string with spaces\n");
      gets(c);   // taking the input string //
      printf("\n String with only one space\n");

      for(i=0;i<strlen(c);i++)
      {
        if(c[i]==' ')// Checks for blank spaces//
        {
            if(inspace==0)
            {
                //replaces multiple space with single space//
                inspace=1;  //change the value of inspace//
                printf("%c",c[i]);
            }
        }
        else//prints the text//
        {
            inspace=0;//change the value of inspace//
            printf("%c",c[i]);
        }
        }

    printf("\n");
    return 0;
}
```

Output:

Run the following commands in your terminal:

gcc space.c

./a.out

```
Enter the sentence with space
I like          programming in      C.
String with only one space
I like programming in C.
```

Aim:

Design, develop and execute a program in C to input N integer numbers in ascending order into a single dimension array, and then to perform a binary search for a given key integer number and report success or failure in the form of a suitable message.

Summary:

> Binary search or half-interval search algorithm finds the position of a specified input value (the search "key") within an array sorted by key value. In each step, the algorithm compares the search key value with the key value of the middle element of the array. If the keys match, then a matching element has been found and its index, or position, is returned. Otherwise, if the search key is less than the middle element's key, then the algorithm repeats its action on the sub-array to the left of the middle element or, if the search key is greater, on the sub-array to the right. If the remaining array to be searched is empty, then the key cannot be found in the array and a special "not found" indication is returned.
> Example:
> Array = 1 3 4 6 8 9 11 Key = 4
> Iteration1:
>
> ```
> Find the mid element of Array , its 6
> Compare key to 6. It's smaller. Repeat with Array = 1 3 4.
> ```
>
> Iteration2:
>
> ```
> Find the mid element , its 3
> Compare key to 3. It's bigger. Repeat with Array = 4.
> ```
>
> Iteration3:
>
> ```
> Find mid element , its 4
> Compare key to 4. It's equal. We're done, we found key.
> ```
>
> In each iteration the length of the list we are looking in gets cut in half. Therefore, the total number of iterations cannot be greater than logN.

Algorithm:

1. Start.
2. Take the input for the size of array (n).
3. Take Input to the array in ascending order.
4. Take the input of key element to be searched(key).
5. Consider low as 0(starting bit address of given array), high as n-1(last bit address of given array).
6. Until low is lesser than or equal to high, do find mid of array, go to step 7.
7. When key is found at mid, go to step 11 , else to step 8.
8. When key is not found at mid go to step 12.
9. When key is found before mid, consider high as mid-1, go to step 6, else to step 10.
10. When key is found after mid, consider low as mid+1, go to step 6.
11. Print successful search with mid value, go to step 13.
12. Print unsuccessful search.
13. Stop.

Program: binsearch.c

```
#include<stdio.h>
void main()
{
    int a[20],i,low,high,mid,n,key,f=0;

    printf("Enter the number of elements\n");
    scanf("%d",&n);   //input for size of the array //
    printf("Enter the elements in ascending order\n");
    for(i=0;i<n;i++)
    scanf("%d",&a[i]); //elements stored in array //
    printf("Enter the key element to be searched\n");
     scanf("%d",&key);  // taking key element to be searched //
    low=0;  // initializing starting address of array //
    high=n-1; // initializing last bit address of array //

    while(low<=high) // loop executes until low=high //
    {
        mid=(low+high)/2;//computing to get mid    position //

        if(a[mid]==key) //executes when key is found at mid //
        {
            f=1;
            break;
        }

        else
        {
            if(key<a[mid]) //executes when    key is found before the mid position //
                high=mid-1;
            else   // executes when key is found after the mid position //
                low=mid+1;

        }
    }

    if(f==1)
          printf("Key element %d is found at position %d\n",key,mid+1);
        //prints when key element found along with    its position//
    else
        printf("Key element %d is not found\n",key);
         // prints when key element not found in the array //
}
```

Output:

Run the following commands in your terminal:

gcc binsearch.c

./a.out

```
1: Enter number of elements
      5
   Enter the elements in ascending order
    1 3 5 7 9
   Enter the Key element to be searched
    7
```

```
  Key element 7 found at position 4
```

gcc binsearch.c

./a.out

```
2: Enter number of elements
    5
   Enter the elements in ascending order
    1 3 5 7 9
   Enter the Key element to be searched
    8
  Key element 7 is not found
```

Aim:

Design, develop and execute a program in C to input N integer numbers into a single dimension array, sort them in to ascending order using bubble sort technique, and then to print both the given array and the sorted array with suitable headings.

Summary:

> Bubble sort technique sorts the integers of the array in ascending order. This technique at first compares the first digit with all the remaining digits individually , if the first number is greater than the other number both the numbers are swapped. Then the second number is compared with all the remaining digits leaving the first digits, if it is greater than other number , both numbers are swapped. This process are done for all the numbers except the last digit. The resulting array of digits will be in ascending order.

Algorithm:

1. Start.
2. Input the size of dimensional array(size=n).
3. Take the input elements to the initialized array.
4. Initialize i=0,j=i+1
5. When a[i]>a[i+1], go to step 6 else to step 8.
6. Swap elements

```
 temp=a[i]
a[i]=a[j]
a[j]=temp
```

7. Increment i and j upto n-1,Go to step 5;
8. Display sorted array.
9. Stop.

Program: bubblesort.c

```
#include<stdio.h>
void main()
{
    int n,i,j,a[10],b[10],temp;
    for(i=0;i<10;i++)
    {
        //for removing junk values in the array //
        a[i]=0;
        b[i]=0;
    }

    printf("\nEnter no of elements\n");
        scanf("%d",&n); //input for size of the array //
    printf("\nEnter the elements\n");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]); // taking array elements //
    for(i=0;i<n;i++)
    {
        b[i]=a[i];           //storing copy of array to an another array //
    }
    //Sorting array using BUBBLESORT method //
    for(i=0;i<n;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(a[i] > a[j])
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
    printf("The given array of elements is \n");
    for(i=0;i<n;i++)
    {
        printf("%d\t",b[i]); //prints the input array//
    }
    printf("\nThe sorted array of given array is\n");
    for(i=0;i<n;i++)
    {
        printf("%d\t",a[i]); //prints the sorted array //
    }
    printf("\n");
}
```

Output:

Run the following commands in your terminal:

gcc bubblesort.c

./a.out

1: Enter no of elements 5

Enter the elements 1 3 5 2 4

The given array of element is 1 3 5 2 4

The Sorted array of the given array is 1 2 3 4 5

gcc bubblesort.c

./a.out

2: Enter no of elements 6

Enter the elements 9 8 6 5 3 4

The given array of element is 9 8 6 5 3 4

The Sorted array of the given array is 3 4 5 6 8 9

Aim:

Design, develop and execute a program in C to compute and print the word length on the host machine.

Summary:

> The word size of a computer generally indicates the largest integer it can process in a single instruction, and the size of a memory address, which is usually, but not necessarily the same as the integer size. The C/C++ standard does not specify the size of integral types in bytes, but it specifies minimum ranges they must be able to hold. The word size can be inferred using predefined architecture/OS/compiler macros provided by the compiler. The below program finds out virtually the word size used by the compiler.
> Lets assume that the word size of the target machine is 8 , (we can consider 16,32 or 64, but we are lazy to write so many zeros in examples)
> So the integer in that machine will be represented with 8 bits, lets use unsigned integer to utilize the sign bit as well.
> now to make an unsigned integer have all 1's in it, we can apply one of the 4 hacks listed below, 1) i = ~0; 2) i = -1; 3) i = (2^8)-1; 4) i= UINT_MAX;
> 1) take negation of 00000000 that is 11111111
> 2) -1 in 2's compilemt is 11111111
> 3) Obvious , but not practical in our case, as we are finding the word lenght, we do not know it beforehand . the value would be 255
> 4) Again not practical in our case, asking the compiler to provide the max value is like cheating right? :P
> Now we have the max unsigned integer variable in our hand, lets destroy every single bit of it , one bit at a time and keep a count of it. when we reach the situation where nothing is left to destroy, the count holds the value of the word lenght
> i=~0 in different word sizes
> bits --> base 10 --> base 2
> 8 --> 255 --> 11111111
> 16 --> 65535 --> 1111111111111111
> 32 --> 4294967295 --> 11111111111111111111111111111111

Algorithm:

```
1) start
2) initialize len=0
3) i=find negation of 0
4) if i is not 0
     go to step5
   else
     got ot step 8
5) i = i>>1
6) len= len +1
7) goto step 4
8) print len
9) stop
```

Program: wordlength.c

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

```
#include<stdlib.h>

int main(void)
{

    int iLen=0;
    unsigned int i=~0;

    for(i = 1; i != 0; i = i << 1)
    {

        iLen++;
    }

    printf("\n\nThe Word length of the host machine is %d\n\n", iLen);
    return 0;
}
```

Output:

Run the following commands in your terminal:

gcc wordlength.c

./a.out

The Word length of the host machine is 32

Aim:

Design, develop and execute a program in C to calculate the approximate value of exp (0.5) using the Taylor Series expansion for the exponential function. Use the terms in the expansion until the last term is less than the machine epsilon defines as FLT_EPSILON in the header file . Print the value returned by the Mathematical function exp ( ) also.

Summary:

> In mathematics, a Taylor series is a representation of a function as an infinite sum of terms that are calculated from the values of the function's derivatives at a single point.
> Exponential formula:
> Sigma (n=0 to infinity) of ((x^n)/n!) where ^ = power , ! = factorial
> (x^0)/0! + (x^1)/1! + (x^2)/2! + ...
> Now obviously we cannot go on till infinity expanding this equation. we have to stop somewhere to get aproximate acceptable answer. We define this termination condition with the fact that our computer has a minimum limit on which the floating operations take place.beyond this limit the values are not representable. This limit is called FLT_EPSILON. The values less than FLT_EPSILON is virtually 0. define FLT_EPSILON 1.19209290E-07F // decimal constant FLT_EPSILON the minimum positive number such that, 1.0 + FLT_EPSILON != 1.0 // do not apply your math here, this is hardware/compiler dependent ;-)
> When the value of one of the component of the above Taylor expression ((x^n) / n!) gets equal to zero, or technically less than FLT_EPSILON we stop the calculation and print the answer.

Algorithm:

1. Start
2. Taylorâ€™s series expansion
   ```
   Let sum=0, x=0.5
   ```
3. Do num=pow(x,i), where value of i=1,2,3,4,5....... (for numerator)
4. Calculate deno=factorial of k, where k ranges from 1 to i (for denominator)
5. Calculate num/den 6. Repeat step 2,3,4 till the value is less than FLT_EPSILON
6. Stop.

Program: Taylor.c

```
#include<stdio.h>
#include<float.h>
#include<stdlib.h>
#include<math.h>

int main(void)
{

        int i,k;
        float X,Sum,Numer,Deno,Fact,Val;

        Sum = 0;
```

```
        X = 0.5;


        for(i=1;i;i++)
        {
                Numer = pow(X,i);
                   /*COMPUTE FACTORIAL*/
                Fact = 1;
                for(k=1;k<=i;k++)
                        Fact *= k;

                Deno = Fact;
                Val = (Numer/Deno);
                Sum += Val;
                if(Val < FLT_EPSILON)
                        break;
        }                /*ADD FIRST TERM IN THE SERIES*/
        Sum += 1;
         printf("\nValue of exp(0.5) after evaluation = %f\n",Sum);

        printf("\nValue of exp(0.5) using built in function = %f\n",exp(0.5));
        return 0;
}
```

Output:

gcc â€“lm Taylor.c

./a.out

```
Value of exp(0.5) after evaluation = 1.648721

Value of exp(0.5) using built in function = 1.648721
```

Aim:

Design, develop and execute a program in C to read two matrices A (M x N) and B (P x Q) and to compute the product of A and B if the matrices are compatible for multiplication. The program is to print the input matrices and the resultant matrix with suitable headings and format if the matrices are compatible for multiplication, otherwise the program must print a suitable message. (For the purpose of demonstration, the array sizes M, N, P, and Q can all be less than or equal to 3)

Summary

> Matrix multiplication, it's a mathematical operations where elements of matrixes are multiplied with other matrix element. The two matrixes are of order (M x N) and (P x Q), for multiplication to exist between matrices the number of columns of first matrix should match the number of rows of second matrix i.e. N=P. Multiplication is carried out by myultiplying row elements of the first matrix with the coloumn elements of the second matrix.

Algorithm

1. Start.
2. Input the order of first matrix matrix1[ ][ ] mxn (m-rows,n-coloumns).
3. Input the order of second matrix matrix2[ ][ ] pxq (m-rows,n-coloumns).
4. When n not equal to p, print a error message and go to step 8.
5. When n=p, perform matrix multiplication, go to step 6.
6. From i=0 to i<m, j=0 to j<q, k=0 to j<p

   ```
   Do    prod[i][j] =Prod[i][j]+ matrix1[i][k] * matrix2[k][j];
   ```

7. Print matrix prod[ ][ ].
8. Stop.

Program: Matrixmul.c

```
#include<stdio.h>
void main()
{
    int i,j,k,matrix1[10][10],matrix2[10][10];
    int m,n,p,q,Prod[10][10];

    printf("Enter the order of the matrix1\n");
    scanf("%d%d",&m,&n);
    printf("Enter the order of the matrix2\n");
    scanf("%d%d",&p,&q);

    //If matrices are not suitable for computation print an   error message //
    if(n!=p)
    {
        printf("Matrix multiplication is not computable for the given matrices\n");
    }

```

```
    else
    {

        printf("\nEnter the elements of Matrix 1\n");

        for(i=0;i<m;i++)
            for(j=0;j<n;j++)
            //taking elements of matrix1//
             scanf("%d",&matrix1[i][j]);

        printf("\nEnter the elements of Matrix 2\n");
        for(i=0;i<p;i++)
            for(j=0;j<q;j++)
            //taking elements of matrix2//
            scanf("%d",&matrix2[i][j]);

        for(i=0;i<p;i++)
            for(j=0;j<q;j++)
              Prod[i][j]=0;

        //To find product and print it if matrices are suitable for computation.//

        for(i=0;i<m;i++)
        {
             for(j=0;j<q;j++)
             {
                  for(k=0;k<p;k++)
                   {
                    // computing product of matrices //
                    Prod[i][j] =Prod[i][j]+ matrix1[i][k] * matrix2[k][j];
                }
             }
        }
               //prints matrix1 //
        printf("\nMatrix 1\n");
        for(i=0;i<m;i++)
        {
             for(j=0;j<n;j++)
             {
                  printf("%d\t",matrix1[i][j]);
             }
            printf("\n");
        }

        //prints matrix2 //
        printf("\nMatrix 2\n");
        for(i=0;i<p;i++)
        {
             for(j=0;j<q;j++)
             {
                  printf("%d\t",matrix2[i][j]);
             }
            printf("\n");
        }

        //to print the product matrix //
        printf("\nThe Product matrix is  \n");
        for(i=0;i<m;i++)
        {
            for(j=0;j<q;j++)
            {
                printf("%d\t",Prod[i][j]);
            }
            printf("\n");
        }

    }

}
```

Output:

gcc matrixmul.c

./a.out

```
1. Enter the order of Matrix1
   3 2
   Enter the order of Matrix2
   4 3
   Matrix Multiplication is not computable for the given matrix
```

gcc matrixmul.c

./a.out

```
2. Enter the order of Matrix1
   3 3
   Enter the order of Matrix2
   3 3
   Enter the elements of Matrix 1
   1 2 3
   4 5 6
   7 8 9
   Enter the elements of Matrix 2
   1 0 0
   0 1 0
   0 0 1
   Matrix 1
   1 2 3
   4 5 6
   7 8 9
   Matrix 2
   1 0 0
   0 1 0
   0 0 1
   The Product matrix is
   1 2 3
   4 5 6
   7 8 9
```

Aim:

Design, develop and execute a parallel program in C to add, elementwise, two one dimensional arrays A and B of N integer elements and to store the result in another one dimensional array C of N integer elements.

Summary:

> The Program add elements of two one dimensional array elementwise and stores it in another one dimensional array.

Algorithm:

1. Start.
2. Take input size of the arrays a[ ] and b[ ].
3. Input the elements to these arrays.
4. Set the number of threads.
5. Until size of reaches maxlength -1
   ```
   Perform addition in parallel by mentioning the
       function within parallel syntax.
   ```
6. Get thread numbers for the operations.
7. Print the resultant array.
8. stop.

Program: parallelprog.c

```
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>

int main()
{
    int a[20],b[20],add[20],i,n;

    printf("Enter the array size\n");
    scanf("%d",&n); // input for size of the array //

    printf("Enter elements of array A\n");
    for(i=0;i<n;i++)
    scanf("%d",&a[i]); //taking elements of array A //

    printf("Enter elements of array B\n");
     for(i=0;i<n;i++)
      scanf("%d",&b[i]); //taking elements of array B //

    printf("Array elements are\n");

    printf("\tArray A\t\tArray B\n");

    for(i=0;i<n;i++)
       printf("\ta[%d]=%d\t\tb[%d]=%d\n",i,a[i],i,b[i]);
       printf("\nComputing Sum...\n");
```

```
    //parallel function to perform array addition //

    omp_set_num_threads(5);
    #pragma omp parallel for private(i)
    for(i=0;i<n;i++)
     {
        int tid=omp_get_thread_num();
        add[i]=a[i]+b[i]; // adding array A and B //
        printf("\nRES[%d]=%d, thread_id=%d",i,add[i],tid);
     }

    printf("\n\nResultant array is\n");

    for(i=0;i<n;i++)
         printf("%d\n",add[i]);//printing resultant array           //

    return 0;
}
```

Output:

Run the following commands in your terminal:

gcc parallelprog.c â€“fopenmp

./a.out

```
Enter the array size
5
Enter elements of array A
1 2 3 4 5
Enter elements of array B
2 3 4 5 6
Array elements are
Array A    Array B
  a[0]=1        b[0]=2
  a[1]=2        b[1]=3
  a[2]=3        b[2]=4
  a[3]=4        b[3]=5
  a[4]=5        b[4]=6

Computing Sum........
RES[4]=11,        thread id=4
RES[1]=5,        thread id=1
RES[2]=7,        thread id=2
RES[3]=9,        thread id=3
RES[0]=3,        thread id=0

The Resultant array is
3
5
7
9
11
```

Aim:

Design and develop a function rightrot(x,n) in C that returns the value of the integer x rotated to the right by n bit positions as an unsigned integer. Invoke the function from the main with different values for x and n and print the results with suitable headings.

Summary:

> Rightrot is a function used to right rotate a integer for specified number of times. This function shifts all the bits of the digit to next bit in right direction and places the LSB(least significant bit) in MSB(most significant bit) position, specified number of times and returns the value.

Algorithm:

1. Start.
2. Take a input number below 65536 to rightrot.
3. Take input for number of times of righrot.
4. Call rightrot function.
5. righrot function;

   from i=0 to n-1

   ```
        When x mod 2 = 0
                Right shift once.
           Else

                Right shift once and add 32768 to value of x
                (I.e. append MSB to 1)
   ```

1. Return the value of x.
2. Stop.

Program: rightrot.c

```
#include<stdio.h>

   // function performing right rotate //
unsigned int right_rot(unsigned int x,int n)
{
        int i;
        for(i=1;i<=n;i++)
        {
                if(x%2==0)
                x=x>>1;
                else
                x=x>>1,x+=32768;
        }
        return x;
}
void main()
{
```

```
        unsigned int x;
        int result,n;

        printf("Enter an integer <=65535\n");
        scanf("%u",&x);

        printf("Rotate %u how many times: ",x);
        scanf("%d",&n);

        result=right_rot(x,n);
        printf("Right_rot(%u,%d)=%u\n",x,n,result);
}
```

Output:

Run the following commands in your terminal:

gcc rightrot.c

./a.out

```
Enter an integer <=65535
28
Rotate 28 how many times
2
Right_rot(28,2)=7
```

gcc rightrot.c

./a.out

```
Enter an integer <=65535
4
Rotate 4 how many times
3
Right_rot(4,3)=32768
```

Aim:

Design and develop a function isprime (x) that accepts an integer argument and returns 1 if the argument is prime and 0 otherwise. The function is to use plain division checking approach to determine if a given number is prime. Invoke this function from the main with different values obtained from the user and print appropriate messages.

Summary:

> A Prime number is a number which can be divisible by 1 and itself, 1 is neither considered as prime nor non-prime(composite). Here a plain division checking approach is used to determine whether a given number is prime or not. The given number is check for divisibility from 1 to itself, if it is not divisible by any other number other than 1 and itself, then that number is a prime number or else it is not a prime number.

Algorithm:

1. Start
2. Take the input integer to be checked
3. Call the isPrime function
4. isPrime function
   ```
        when value=1, print it is niether prime nor             composite
        from i=2 to value perform x (mod) i
            when x%i is equal to 0,  return 0 else     return 1
   ```
5. when value return is 1, print number is prime else step 6.
6. i.e. The return is 0, print number is not prime.
7. Stop

Program: isprime.c

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

int isPrime(int);

int main(void)
{
        int val,flag;

        printf("\nEnter the value to be checked\n");
        scanf("%d",&val); //Taking input//

        flag = isPrime(val);//Calling isprime function//

        if(flag==1)  //Prints if the number is prime//
          printf("\nThe entered value %d is a prime number\n",val);

        else   //Prints if the number is not prime//
          printf("\nThe entered value %d is not a prime number\n",val);

```

```
        return 0;
}

int isPrime(int x)
{
        int i;
        if(x==1)  //Executes if the given value is 1//
        {
              printf("\n1 is neither prime nor composite\n");
          exit(0);
        }
        for(i = 2; i<x ;i++)
        {
                if(x % i==0)// Checks for divisibility//
                return 0;
        }
        return 1;
}
```

Output:

Run the following commands in your terminal:

gcc isprime.c

./a.out

```
Enter the value to be checked
1
1 is neither prime nor composite
```

gcc isprime.c

./a.out

```
Enter the value to be checked
27
The entered value 27 is not a prime number
```

gcc isprime.c

./a.out

```
Enter the value to be checked
29
The entered value 29 is a prime number
```

Aim:

Design, develop and execute a parallel program in C to determine and print the prime numbers which are less than 100 making use of algorithm of the Sieve of Eratosthenes.

Summary:

Algorithm:

1. Start
2. Input the value of n, from i=2 upto n store the value in an array.
3. Set the number of threads.
4. From k=2 to sqareroot of n
   ```
   When array element at k is not 0
   For all multiples of k*k assign 0, increment k.
   ```
5. From i=2 to n,
   ```
   Print the numbers of array which is not 0.
   ```
6. Stop.

Program:

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

int main(void)

{
    int a[200];
    int n,i,k;

    printf("Enter the value of n:\n");
    scanf("%d",&n);  // Input value//

    for(i=2;i<=n;i++)
    {
        a[i]=i;  //Storing the number into array upto n//
    }

    omp_set_num_threads(5); //Setting up threads//
    for(k=2;k<=sqrt(n);k++)
    {
            if(a[k]!=0)
            {
        #pragma omp parallel for private(i)
        for(i=k*k;i<=n;i=i+k)  //Assigning 0 to multiples//
        {
              printf("\nThread id=%d makes %d position zero",omp_get_thread_num(),i);
                a[i]=0;
            }
```

```
            }
    }
    printf("\nPrime numbers are \n");
    for(i=2;i<=n;i++)
    {
        if(a[i]>0)
         printf("%d\t",a[i]); //Printing prime numbers//
    }

    printf("\n");

    return 0;
}
```

Output:

gcc prime.c â€“fopenmp â€“lm

./a.out

```
Enter the value of n:  25

Thread id=0 makes 4 position zero

Thread id=3 makes 22 position zero

Thread id=3 makes 24 position zero

Thread id=0 makes 6 position zero

Thread id=0 makes 8 position zero

Thread id=1 makes 10 position zero

Thread id=1 makes 12 position zero

Thread id=1 makes 14 position zero

Thread id=2 makes 16 position zero

Thread id=2 makes 18 position zero

Thread id=2 makes 20 position zero

Thread id=0 makes 9 position zero

Thread id=0 makes 12 position zero

Thread id=1 makes 15 position zero

Thread id=1 makes 18 position zero

Thread id=2 makes 21 position zero

Thread id=2 makes 24 position zero

Thread id=0 makes 25 position zero

Prime numbers are
```

```
2    3    5    7    11    13    17    19    23
```

Aim:

Design and develop a function reverses (s) in C to reverse the string s in place. Invoke this function from the main for different strings and print the original and reversed strings.

Summary:

> Many operations on strings can be performed, reversing the string is one of the operations which can be performed. At first the given string is copied to another empty temporary string, then in the temporary string, first character of the string and the last character of the string is swapped or interchanged, then the second character and last but one character is swapped, this process is done until the middle element of the string (In case if the string contains even number of characters , process is done until the middle two elements). Then the reversed string and the original string are printed.

Algorithm:

1. Start.
2. Take an input string.
3. Compute the input string length.
4. Call the reversing function.
5. Initialize variable i=0,and j=string length(input)-1
6. until i is less than j, go to step 6, else go to step 7

```
   Swap characters
   temp=a[i];
   a[i]=a[j];
   a[j]=temp;
```

7. Stop.

Program:reversestring.c

```
#include<stdio.h>
#include<string.h>
void reverses(char *);

void main()
{
    char a[100];

    printf("Enter the string to be reversed\n");
    scanf("%s",a); // reading string //

    printf("The original string is %s\n",a);
    reverses(a); //calling a function //

    printf("The reversed string is %s\n",a);
}

void reverses(char *a) // function to reverse a string //
{
    char temp;
    int i,j;

    i=0; //initializing i to start bit of string //
    j=strlen(a)-1; //initializing j to last bit of string //


    while(i<j)
    {    //reversing the string //
        temp=a[i];
        a[i]=a[j];
        a[j]=temp;
        i++;
        j--;
    }
}
```

Output:

gcc reversestring.c

./a.out

```
1. Enter the string to be reversed
   fsmk

   The original string is
   fsmk

   The reversed string is kmsf
```

Aim:

Design and develop a function matchany (s1,s2) which returns the first location in the string s1 where any character from the string s2 occurs, or â€“ 1 if s1 contains no character from s2. Do not use the standard library function which does a similar job! Invoke the function matchany (s1,s2) from the main for different strings and print both the strings and the return value from the function matchany (s1,s2).

Summary:

> The matchany function returns the first location of the character of the second string if it is present in the second string, or else it returns -1. It first checks the first character of the first string with all the characters of the second string(from first to last of second character), if any matches is found it returns the location of that character in the first string, else it searches for the second character , same process is carried on till the last character of the first string if any matches are not found in between.

Algorithm:

1. Start
2. Take the input strings
3. Call the function matchany
4. From i=0 to string length, check whether the character in first string matches with second string.
5. When match found return the position of character in first string,else exit.
6. Stop

Program: matchany.c

```
#include<stdio.h>
#include<string.h>
int matchany(char *,char *);
void main()
{
    char a[100],b[100];
    int f;
    printf("Enter the first string\n");
    scanf("%s",a);
    printf("Enter the second string\n");
    scanf("%s",b);

    f=matchany(a,b);//calling a function //

    //printing two strings //
    printf("The First string is\n %s\n",a);
    printf("The Second string is\n %s\n",b);


    if(f==-1)  // executes when string does not match //
     printf("Character did not match\n");
    else     //prints the matching string with its position //
     printf("The character %c of the second string is found at position %d of first string\n",b[f],f+1);
}

int matchany(char *a,char *b)   // function to find a match  pattern in string //
{
    int i,j;
    for(i=0;i<strlen(a);i++)
    {
        for(j=0;j<strlen(b);j++)
        {
        if(a[i]==b[j])
        return i;
        }
    }
    return -1;
}
```

Output :

gcc matchany.c

./a.out

```
    1. Enter the first string
       book
       Enter the second string
       pencil
       The First string is
       book
       The Second string is
       Pencil

       Character did not match
```

gcc matchany.c

./a.out

```
    2. Enter the first string
       computer
       Enter the second string
       machine
       The First string is
       computer
       The Second string is
       machine

       The character c of the second string is found at position 1 of first string
```