RECURSION – IV

Finding square root of a number
Generating Permutations
Additive sequence

Finding Square Root of a number:

We can use the Newton's method, which updates a given guess. Let us take an example. Let the given number be 36. Let us guess that its root is 9. Now since 36/9 is 4, it is certain that the new value of the root has to lie between 9 and 4, as their product equals the given number. So the best way to make the next guess would be to choose a number lying somewhere in the middle of the range (4, 9). This would be (4 + 9)/2, i.e. 6.5 which is more closer to the actual root 6 than the first guess 9 was. We can proceed further by computing the next factor 36/6.5 and then taking the mid value of this factor and 6.5. We see that we very rapidly approach 6, the true value of the root. The steps can be repeated till we find that the current step did not produce appreciable change in the value of the root.

Thus given a number x, the first guess could be r = x/2.
We keep on generating and testing successive approximations of r until we find the value that is close enough to stop. The updating process may be terminated when
| r * r – x | < epsilon.
A recursive solution to this problem would need the routine to be called repeatedly with the new value of the root. The stopping condition for the recursive function is already indicated above.

```
#include <stdio.h>
double sqroot(double num, double root);

int main ( )
{
 double x,r;
printf("\n enter x=");
scanf("%lf", &x);
r = x/2;
r = sqroot(x,r);
 printf("\n root =  %10.2f",r);
printf("\n");
}
double sqroot(double num, double root)
{
  double newroot,error;
  error = root*root - num;

  if ( abs(  error )<= 0.00001)
        return root;
  else {
        newroot= 0.5*(root + num/root);
        return sqroot(num,newroot);
          }
}
```

Generating Permutations:

Permutations are often needed in games like Scrabble, where we want to consider all possible rearrangements with a given set of letters.

Different permutations of a given string can be generated by keeping the first letter fixed and recursively finding permutations of remaining letters. Then the first letter can be exchanged with one of the other letters and the process repeated.

Consider the string CD.
It has only two permutations obtained by exchanging the characters :
CD
DC

Any string of 2 characters would have 2 possible permutations.

Next, consider the string BCD.
Keep B fixed and permute CD to yield
BCD
BDC

Get back to original sequence BCD. Then exchange B with C to yield CBD.
Now keep C fixed and permute BD to yield
CBD
CDB

Get back to original sequence BCD. Finally exchange B with letter D to yield DCB.
Now keep D fixed and permute CB to yield
DCB
DBC

Thus a string of 3 characters has 6 possible permutations. A string of 4 characters has 12 possible permutations. For a string of length n , the number of permutations are n!

To generate all possible permutations for a string of n characters the following recursive algorithm can be used:
Keep the first k letters fixed and permute the remaining. Start with k = 0 and permute the remaining. For each permutation, increase k and continue to permute using a *for* loop. Print the characters when k equals n.

•The C prototype for this function :