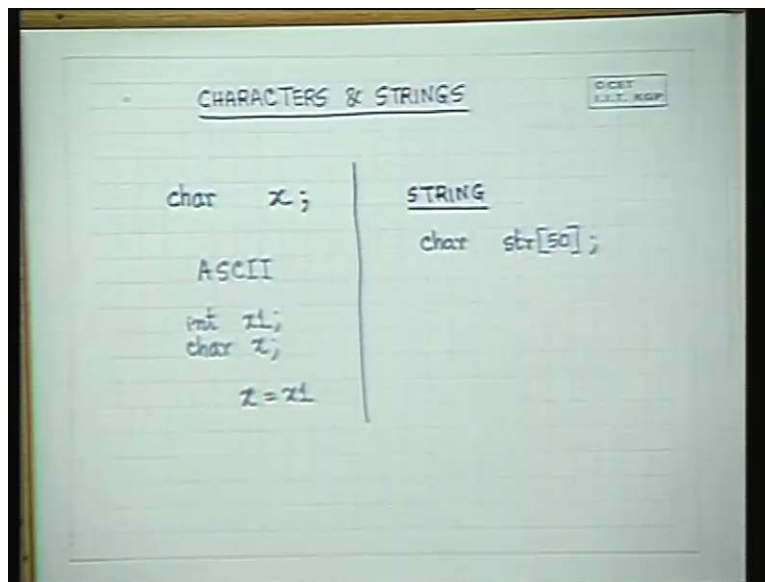


Programming & Data Structures
Dr. P.P. Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 12
Characters and Strings

The topic of today's lecture is a characters and strings. In C, we have till now just had a quick look at integers, floating point numbers and arrays. Characters can be declared and we have possibly seen in 1 or 2 examples before by the declaration `char x`.

(Refer Slide Time: 03:08)

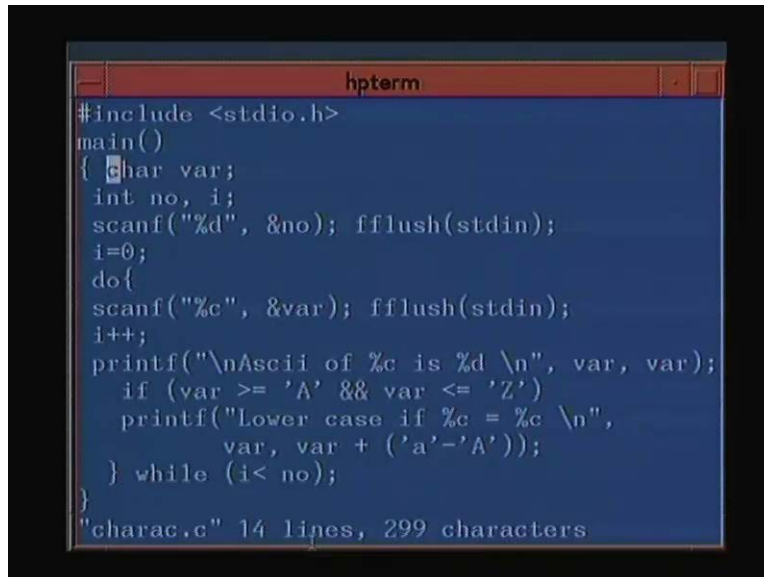


This is equivalent to a single character. Characters in C are stored in ASCII format and in C all characters are stored in ASCII format and inter conversion from ASCII format to integers is very easy and we can always assign an array to an integer. So if you have an `int x` or `int x1` and then `char x`, if you say `x` is equal to `x 1` then this will now contain the ASCII value and represent the ASCII value of the character. So instead for characters we will just see one simple program which will help us to understand how characters are declared and how the ASCII values can be printed and in what format characters can be printed.

After we see characters, the next we thing that we will see is a string. A string is an array of character and we can declare it as an array of characters like and if you store this element of this string in a proper format then you can use several string operations which are functions which are provided in the C library for comparing strings, for copying strings and for various other purposes.

So let us see first an example on the use of characters and then we will move on to strings. This is an example in which we are declaring a character, we are reading in a character and after reading in a character we are printing the character, its ASCII value and then if it's an upper case then only we are converting it into a lower case character.

(Refer Slide Time: 03:45)

A screenshot of a terminal window titled 'hpterm' with a blue background. The window displays C code for a program named 'charac.c'. The code includes <stdio.h>, defines a main function, declares a character variable 'var' and integer variables 'no' and 'i'. It uses scanf to read 'no', then enters a loop that reads characters with scanf, prints their ASCII values, and converts uppercase characters to lowercase. The code ends with a while loop condition 'i < no'. At the bottom of the terminal, it shows the file size: 'charac.c' 14 lines, 299 characters.

```
#include <stdio.h>
main()
{ char var;
  int no, i;
  scanf("%d", &no); fflush(stdin);
  i=0;
  do{
  scanf("%c", &var); fflush(stdin);
  i++;
  printf("\nAscii of %c is %d \n", var, var);
  if (var >= 'A' && var <= 'Z')
  printf("Lower case if %c = %c \n",
        var, var + ('a'-'A'));
  } while (i< no);
}
```

"charac.c" 14 lines, 299 characters

So this is the main program which starts here and ends here. Is it visible? We declare a character variable called var, char var and we declare two integers no and i. No is the number of characters that we will read in and i is the loop control variable. First is we scanf and number which means we read in the number of characters which we will input. The next function is an interesting command called fflush stdin. Now fflush stdin is there, there are two buffers one is for the output and one is for the input maintained in C. So if you just start cleaning, data is automatically stored in these buffered locations. Buffer can be portion of the memory where your inputs are stored in.

Now when we enter AND number, we will be entering the character and then pressing the enter command. So both the character and the enter command will be stored here and unless we flush this for the next scanf which occurs the entire character will be taken in. That is why we flush this one to clear out the input buffer. There are other commands to read in characters called getchar which you can have a look at the books and we will also see in some examples later on. But first let us see how we will use it using scanf.

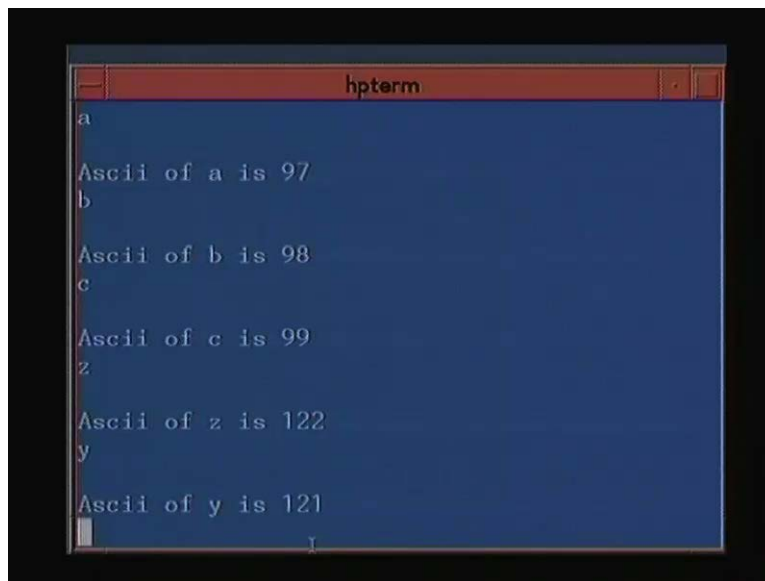
So we first read the number of numbers initialize i to 0 and in a loop we read in a no that is no that is number of characters and for each character we print the character, we print its ASCII value and if it's an uppercase character, we print the lower case. So we read in a character using a % c format. In % c you can read in a single character, so we read in character and % c format, increment i and print the ASCII value of the variable c.

So we can read and print in % c format and for the same variable if you give the % d format then you will get it in decimal and you can see the ASCII value of the variable. So that is why we have put it in var here and again var. In the first case we printed in % c format and in the second case to get this ASCII value we print it in % d format. And you can use such variables and compare characters with characters or character in var we will compare with A but you must give it in quotation marks. It will also compare less than equal to z but you must give it in quotation marks. What happens is the ASCII values are compared. They are just simply compared like integers in ASCII terms and equality, not equality, greater than equal to less than equal to all these comparisons can be done. Also in the ASCII format the characters small a to small z are in contiguous, have contiguous values.

Similarly capital a to capital z also have got contiguous values. So if want to check whether it is either a or b or c or d or up to z then it is sufficient to check whether the variable is greater than equal to A and the variable is less than equal to Z. And we will obtain the lower case, how? The smaller the ASCII values of the lower cases are greater than the ASCII values of the upper cases. So you find the difference between the lower case and the upper case ASCII values and you add it to the ASCII values of the variable then immediately the upper case will be converted to the lower case.

For example you will see, let us see one or two examples it will be very clear to us what we are doing. So let's run the program.

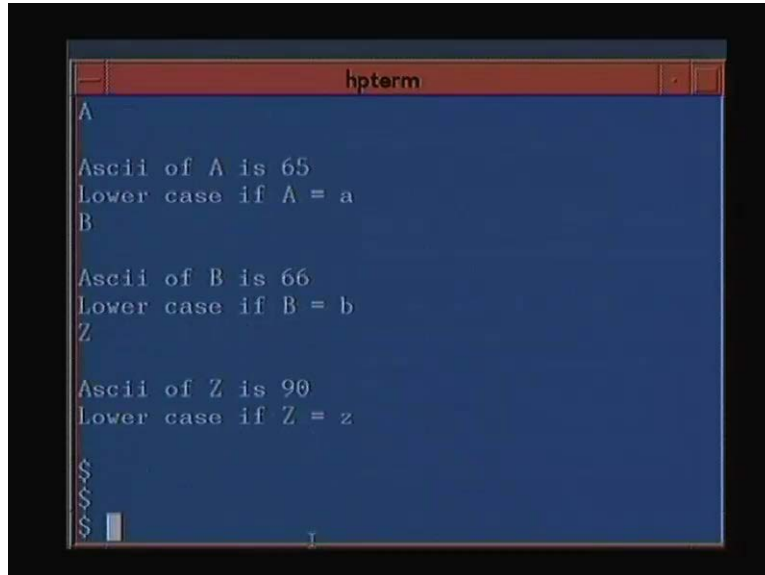
(Refer Slide Time: 08:21)



```
hpterm
a
Ascii of a is 97
b
Ascii of b is 98
c
Ascii of c is 99
z
Ascii of z is 122
y
Ascii of y is 121
```

So let us give in some 10 numbers we will input and let us first give in only small lower case characters to see what the ASCII values are. a the ASCII value is 97, b 98, c 99, z 122, y 121. So these are the ASCII values from 97 to 122 is the lower case characters. Upper case a is 65 and we have also found the lower case it will be seen from these characters.

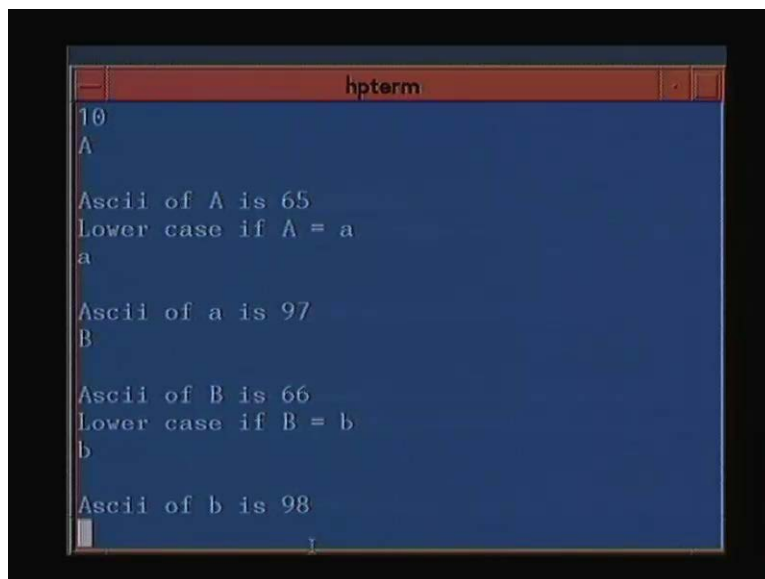
(Refer Slide Time: 08:45)



```
hpterm
A
Ascii of A is 65
Lower case if A = a
B
Ascii of B is 66
Lower case if B = b
Z
Ascii of Z is 90
Lower case if Z = z
$
$
$
$
```

A is 65, B is 66 and Z is 90, so let's go back to the program. So at this point however we found converted from upper case to lower case. We have seen the upper case suppose it is 60 and then we find the difference between the ASCII values of the lower case. You can do a subtraction addition because subtraction and addition here will be done in terms of ASCII values. So you just do a subtraction that is you find out the shift between the lower case and the upper case and you just add it to the uppercase ASCII value, you automatically get the lower case where you printed in % c format. So this is how we got the lower case if A is 65, small a is 97, take b and take small b so the difference you will get is 1 and then you just add it up.

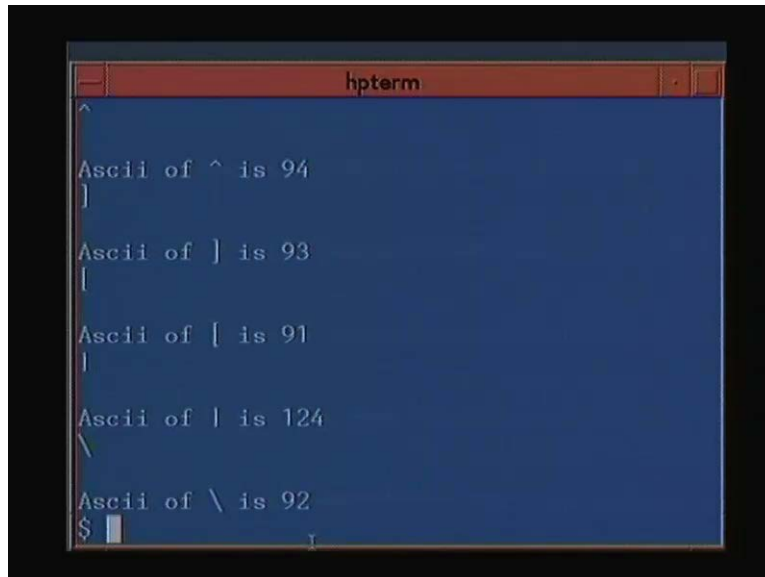
(Refer Slide Time: 09:45)



```
hpterm
10
A
Ascii of A is 65
Lower case if A = a
a
Ascii of a is 97
B
Ascii of B is 66
Lower case if B = b
b
Ascii of b is 98
```

The difference between lower case a and upper case A is 97 minus 65 that is 32 and you just add 32 to 66 which will give you 98. So this is how you get the ASCII values. Actually you can get ASCII values of various other characters also bracket closed, look at this 94.

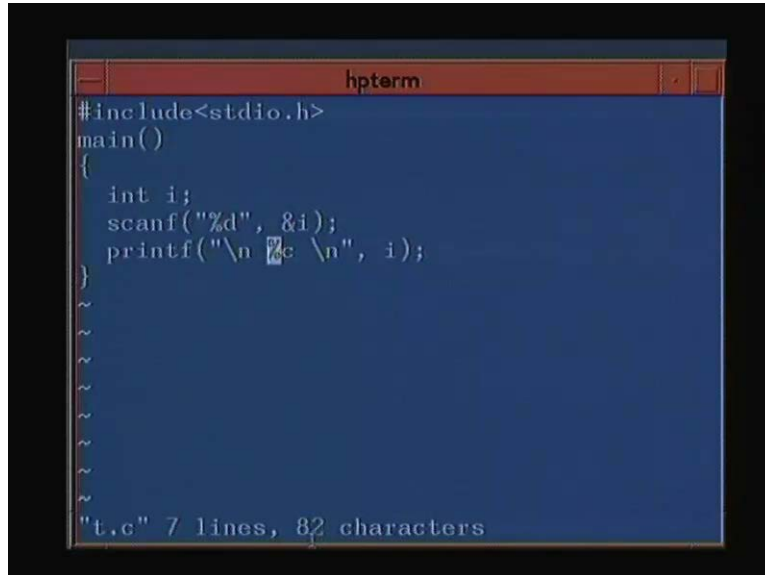
(Refer Slide Time: 10:32)



```
hpterm
^
Ascii of ^ is 94
]
Ascii of ] is 93
[
Ascii of [ is 91
|
Ascii of | is 124
\
Ascii of \ is 92
$
```

There was a gap between 90 and 97, see this one is 94, there will be others I think this one will give you 93, this one will give you 91 and you can get several characters all these. Then beyond 122 you have got some characters, see this one is 92 and so on. And if you want to know what is the character value of an ASCII variable, you can just write a simple program which I just wrote. You just scanf the number in % d format and then print it in % c format and you will easily get the ASCII value.

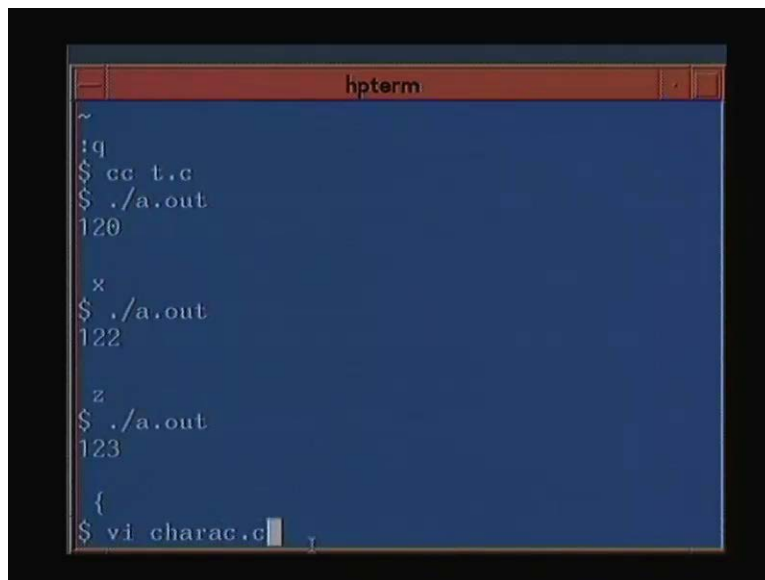
(Refer Slide Time: 10:48)



```
hpterm  
#include<stdio.h>  
main()  
{  
    int i;  
    scanf("%d", &i);  
    printf("\n %c \n", i);  
}  
~  
~  
~  
~  
~  
~  
~  
~  
~  
"t.c" 7 lines, 82 characters
```

Suppose we want 120 that's x, suppose we want 122 that's z, we want to go somewhere beyond that say 123 which is bracket open and you can find out the ASCII values of various elements.

(Refer Slide Time: 11:22)



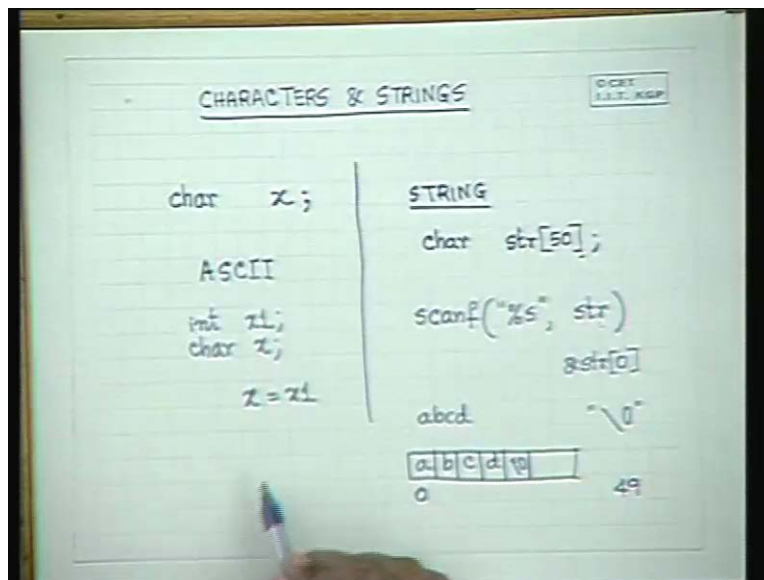
```
hpterm  
~  
:q  
$ cc t.c  
$ ./a.out  
120  
  
x  
$ ./a.out  
122  
  
z  
$ ./a.out  
123  
  
{  
$ vi charac.c
```

So there are two things we learn here, few things we are learn here one is the declaration of characters variables, one is the concept of fflush stdin, you may also require fflush stdout in other cases but we normally don't do it because we use that back slash n and other things then we print the full buffer but when we start using files and do other things we will see this flushing out may be very very important. We will also learn the % c

format, the ASCII in automatic inter conversion from % d to % c and arithmetic operations can be done on characters individual characters and this arithmetic operation will do it on the ASCII values. So this is a very simple exercise on character, next we will tackle strings.

As I mentioned earlier to define a string, a string is a character array. So you can define it to be a character array and use it as a string and C provides you certain facilities to directly read and write strings. And if you want to read a string, if you want to read the string str we have to give scanf % s format.

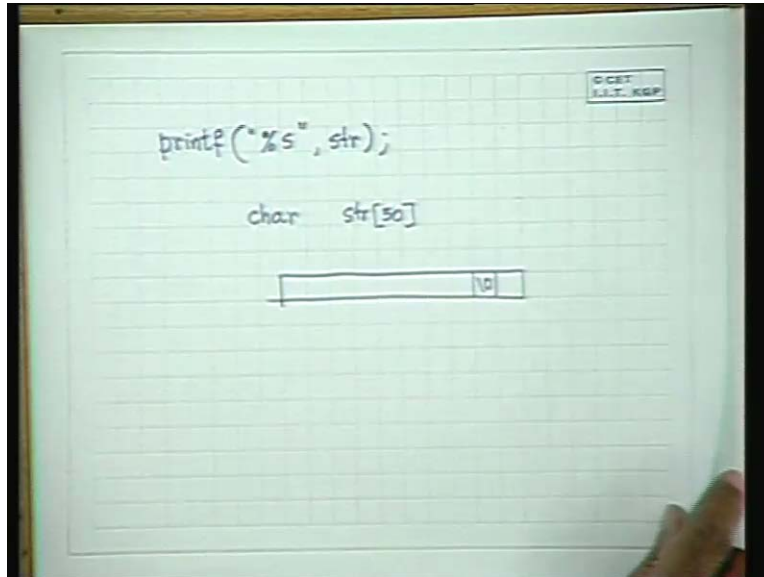
(Refer Slide Time: 14:25)



In % s format we can read the string str and obviously you can just simply give the name of the string. You can give the name of the string, the name of the string is sufficient to provide you the address of the string which is the same as and str [0]. We will arrays and indices and addresses to find out. So you have to give the starting of the string or the address of the string which is equivalent to giving the name of the string and once you give the name of the string, automatically the string will be read and if you write down a b c d then it will store it in the array like this a b c d and it will end it with a special symbol which is called back slash zero.

It will end it with this input. The string, all strings are ended with an end marker like this. So if you declare a string of size 50 which means an array from 0 to 49 make sure that your string does not exceed this size, so that the back slash zero cannot fit in. The backslash zero must be put into the string. Similarly like scanf you have, for strings you can use the printf command.

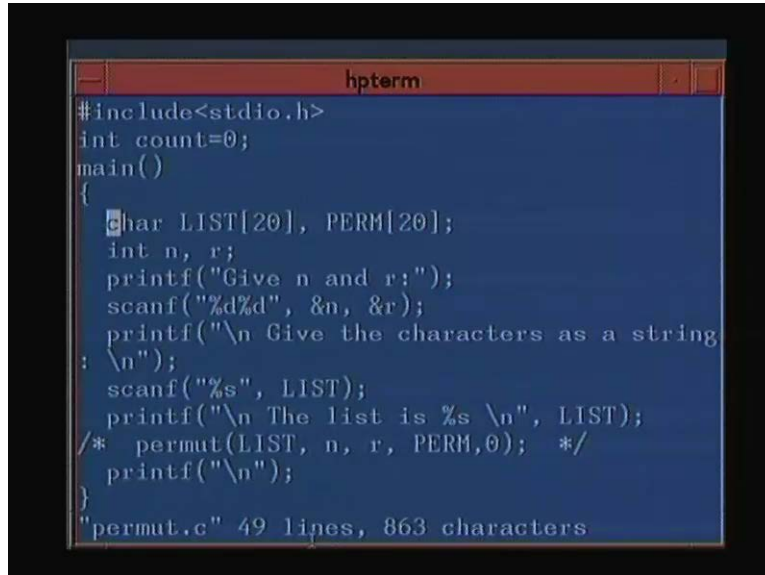
(Refer Slide Time: 15:40)



Printf % s, again you just give the string, it will print the string but it will print a string which is whatever address you give this must be defined as a character array, whatever you print in % s format must be defined as a character array firstly and secondly the data must end with a backslash zero. If there is no backslash zero then this function for **printing f**, printing in % s format will go on continuing till the end, alright.

So we will just see how to read and write strings and then we will work out the permutations problems which we were discussing on these students. So we will just look at the permutations program for just reading and writing a string and then we will see how we actually do the permutations and how we write the program for permutations. This is the program where this is the main program and we will not call the permutation, we will just look at strings first. Here I have declared two character arrays or you can look at a character array as a string.

(Refer Slide Time: 16:30)

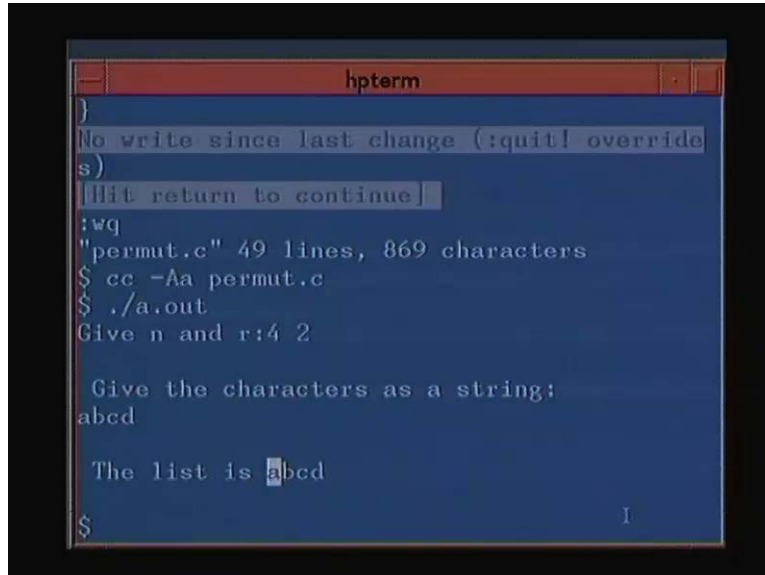


```
hpterm
#include<stdio.h>
int count=0;
main()
{
    char LIST[20], PERM[20];
    int n, r;
    printf("Give n and r:");
    scanf("%d%d", &n, &r);
    printf("\n Give the characters as a string
: \n");
    scanf("%s", LIST);
    printf("\n The list is %s \n", LIST);
    /* permut(LIST, n, r, PERM,0); */
    printf("\n");
}
"permut.c" 49 lines, 863 characters
```

One is list 20 of size 20, one is perm, perm of size 20 two integers n and r and in a permutations problem, you will first give n and r and read it as AND n and AND r and then we are giving the characters that is the n characters, we will input as a string okay. So we say printf give the characters as a string and then we read it in a % s format into the character array list. So we read it like this % s and then we can print it in % s format like this. There is a call to permutation which I have commented out for the time being. I have purposely commented out, we will have a look at it much later on. So comments can be given like this slash star and ended up with star slash.

So this program will not execute this part, this will not be compiled, so we will just have a look at reading and writing strings. Give n and r that is alright 4 and 2 and then we have to give characters, so we will give a b c d, like this you can give the string it will read it and it will print it, the list is a b c d.

(Refer Slide Time: 18:08)



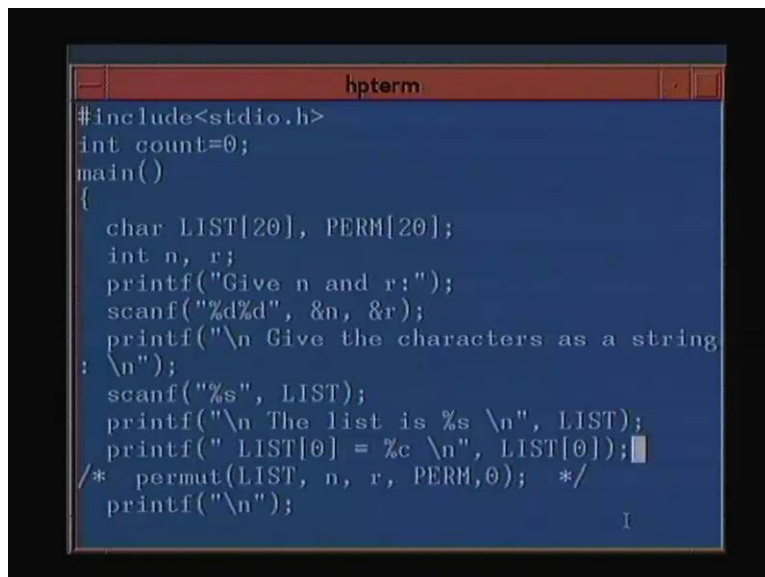
```
hpterm
}
No write since last change (:quit! override
s)
[hit return to continue]
:wq
"permut.c" 49 lines, 869 characters
$ cc -Aa permut.c
$ ./a.out
Give n and r:4 2

Give the characters as a string:
abcd

The list is abcd
$
```

So using the % s format you can read and write a string. You can also use, you can also use it as a character array and tackle it with individual characters, we can use list as an individual character array and you can, if you want you can print out. Suppose you wanted to print out in % c format say LIST [0] equal to % c and you want to print out the value of LIST [0] that also you can do and print it out as a character.

(Refer Slide Time: 18:58)

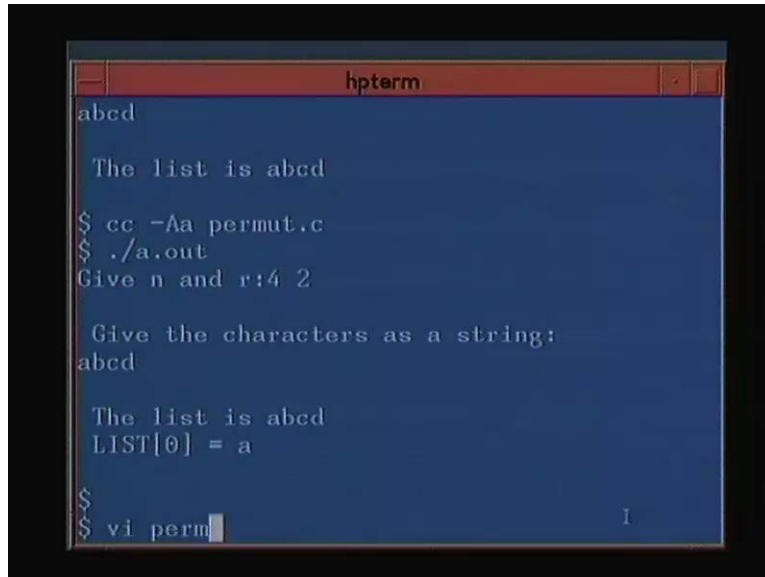


```
hpterm
#include<stdio.h>
int count=0;
main()
{
char LIST[20], PERM[20];
int n, r;
printf("Give n and r:");
scanf("%d%d", &n, &r);
printf("\n Give the characters as a string
: \n");
scanf("%s", LIST);
printf("\n The list is %s \n", LIST);
printf(" LIST[0] = %c \n", LIST[0]);
/* permut(LIST, n, r, PERM,0); */
printf("\n");
}
```

So let us see how this... Oh, I didn't compile it, I am sorry I didn't compile it, so the previous version will...

So we got list is a b c d and LIST [0] is a and if you want to see how the list is ended then suppose we give 4 and if you want to print out if list 4 we will see that there will be a backslash zero marking the end of the list.

(Refer Slide Time: 19:45)



```
hpterm
abcd

The list is abcd

$ cc -Aa permut.c
$ ./a.out
Give n and r:4 2

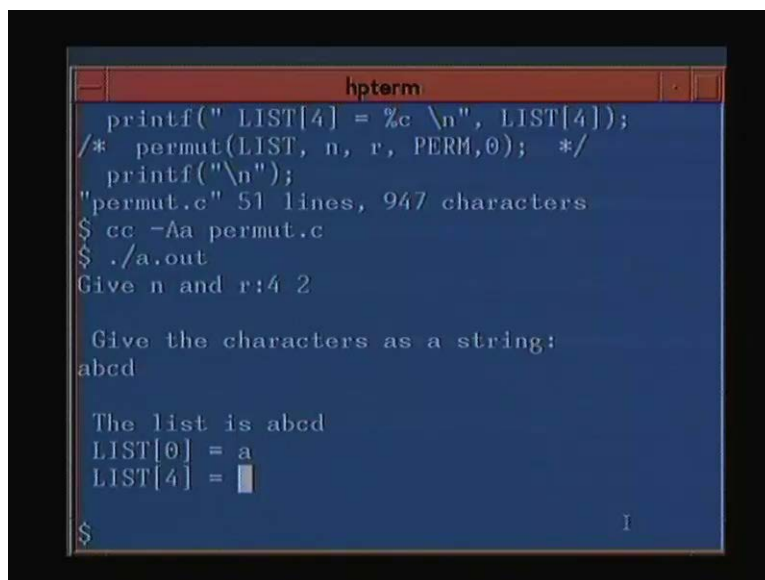
Give the characters as a string:
abcd

The list is abcd
LIST[0] = a

$
$ vi perm
```

0 1 2 3 will be a b c d and list 4 will be the end of the list. Here backslash zero could not be printed but I don't know why but this should be backslash zero.

(Refer Slide Time: 20:30)



```
hpterm
printf(" LIST[4] = %c \n", LIST[4]);
/* permut(LIST, n, r, PERM,0); */
printf("\n");
"permut.c" 51 lines, 947 characters
$ cc -Aa permut.c
$ ./a.out
Give n and r:4 2

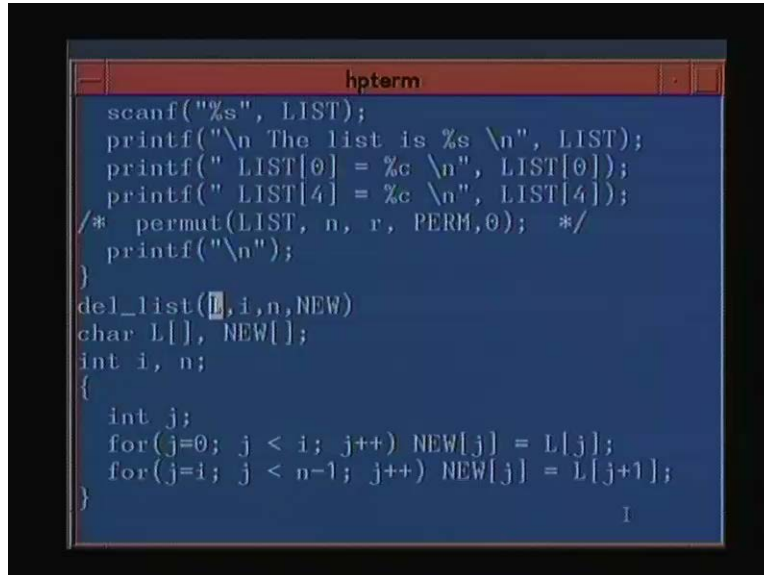
Give the characters as a string:
abcd

The list is abcd
LIST[0] = a
LIST[4] = █

$
```

This is... your blank is printed because I think it is not an ASCII character, it is not one of those it is a special character. So it is the backslash zero which is there in list form.

(Refer Slide Time: 21:15)

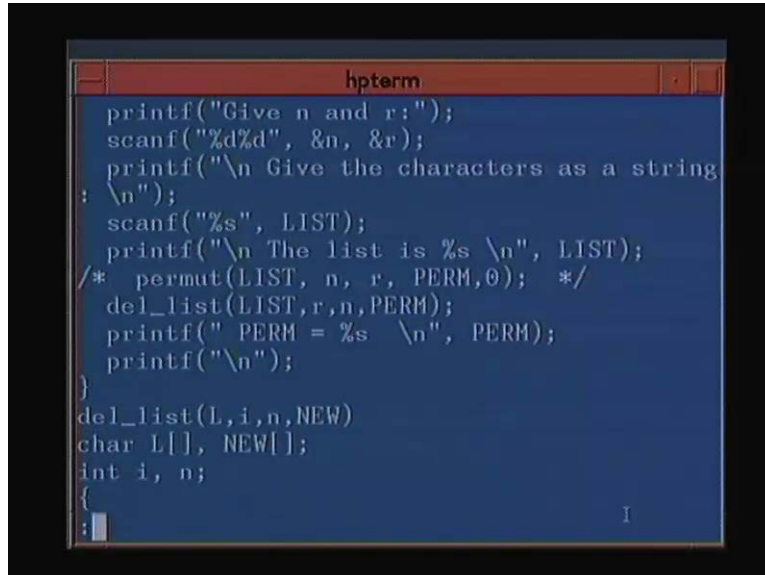


```
scanf("%s", LIST);
printf("\n The list is %s \n", LIST);
printf(" LIST[0] = %c \n", LIST[0]);
printf(" LIST[4] = %c \n", LIST[4]);
/* permut(LIST, n, r, PERM,0); */
printf("\n");
}
del_list(L,i,n,NEW)
char L[], NEW[];
int i, n;
{
    int j;
    for(j=0; j < i; j++) NEW[j] = L[j];
    for(j=i; j < n-1; j++) NEW[j] = L[j+1];
}
```

So now let us go back to the program to see a function on lists. Here a function, we have written a function delete list. Given a character array or a string L and a position i and there are n elements in the string, NEW will be the second character array. And what will happen is here in the string whatever is given as L , the i 'th element will be deleted and the rest of the string will be returned in NEW . So how do we do that? We declare a loop control variable, for j equal to 0 till j is less than i we copy $L[j]$ into $NEW[j]$ and we just leave out i minus 1, sorry we leave out i minus 1 because the i 'th element if the array starts from 0 then the i 'th element will be $L[i - 1]$.

So the rest of it we do from i till $n - 1$ because one element less. And we copy into $NEW[j]$, we have done it up to $i - 1$ isn't it and we have to remove out the i 'th element. Am sorry we start from 0, I will just repeat once more. We have to remove the i 'th element, so up to $i - 1$ we copy into $NEW[j]$ and for the rest of the elements we copy $L[j + 1]$ into $NEW[j]$.

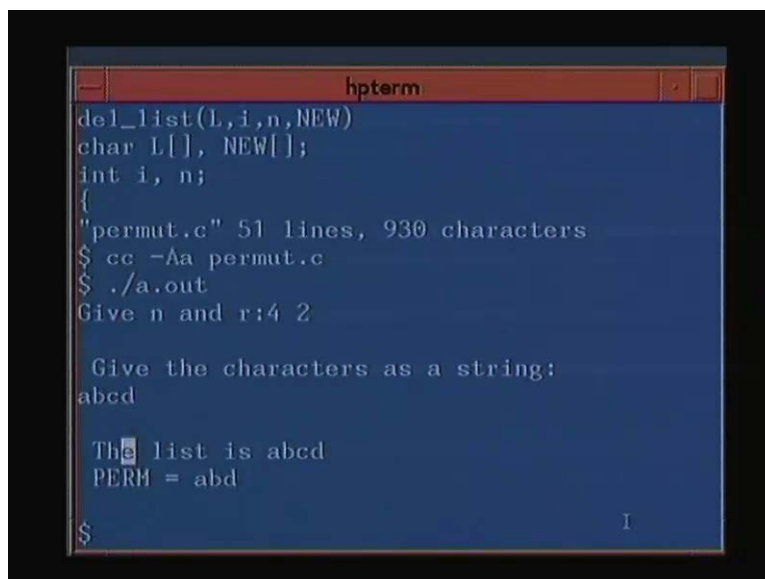
(Refer Slide Time: 23:55)



```
hpterm
printf("Give n and r:");
scanf("%d%d", &n, &r);
printf("\n Give the characters as a string
: \n");
scanf("%s", LIST);
printf("\n The list is %s \n", LIST);
/* permut(LIST, n, r, PERM,0); */
del_list(LIST,r,n,PERM);
printf(" PERM = %s \n", PERM);
printf("\n");
}
del_list(L,i,n,NEW)
char L[], NEW[];
int i, n;
{
:
I
```

So that way one element will be removed and if you call it from here that is delete these and after having read list, suppose we call the list and we have to call it with LIST, I say let us call it with 3, you can enter i but we are purposely giving 3 or let us give r because we have read in a variable r, m and let us take in the other character array perm which we have % s. So we have read it into, so we have read in a list we have read in n and r and we have removed the r'th element and stored it in an another string perm and we have just printed that. So this is the delete list function, so this is a b c d we read in a b c d which is in 0 1 2 and 3 and we removed the second for i equal to 2 and we get back abd.

(Refer Slide Time: 24:16)



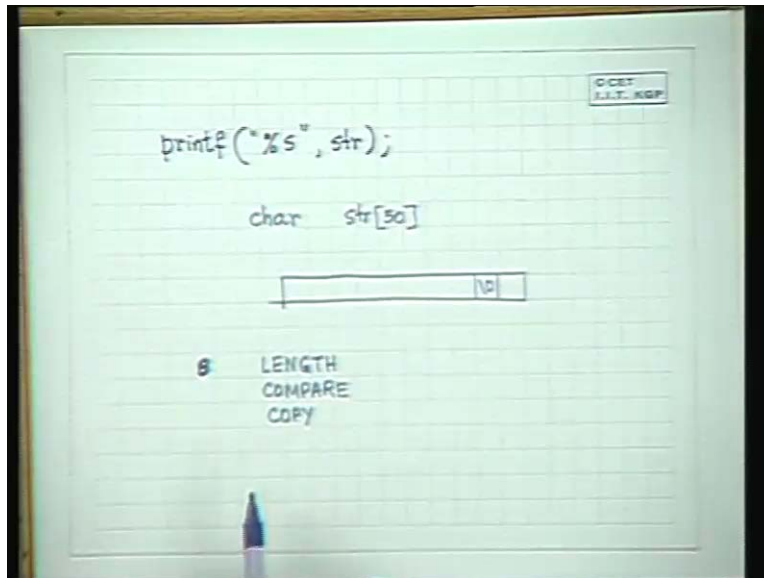
```
hpterm
del_list(L,i,n,NEW)
char L[], NEW[];
int i, n;
{
"permut.c" 51 lines, 930 characters
$ cc -Aa permut.c
$ ./a.out
Give n and r:4 2

Give the characters as a string:
abcd

The list is abcd
PERM = abd
$
I
```

So this is how we can use, we can manipulate character arrays and use the string functions. There are several other string functions which are available from the library there is a string library also and there is a normal library and you can find out things like length of a string. You can compare strings and you can copy strings and **there are several** you can duplicate there is a string copy, there is a string compare, various types of string compare commands, there is a string length command.

(Refer Slide Time: 25:18)



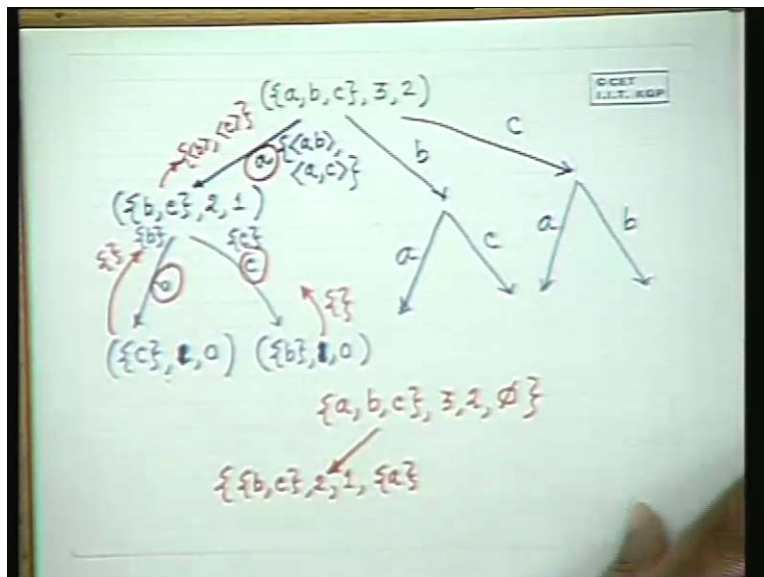
We can quickly have a look at the library to see what these commands are. So finally we will now go to our permutations program and see how we can finally now implement the permutations program. Among the various versions of permutations program, we pick up this version where in the loop we delete one element at a time and then recursively call it and come back.

(Refer Slide Time: 26:46)

```
List permut(L, n, r)
char L[];
int n, r;
{
  if (r == 0) return {};
  else {
    for (i = 0; i < n; i++) {
      del_list(L, i, n, NEW);
      RES = permut(NEW, n-1, r-1);
      FINAL = append_in(RES, L[i]);
    }
    return (FINAL);
  }
}
```

So list permut L, n and r, l is a character array or a string, n and r are integers. If r is equal to 0 we returned a null string, we return a null. This returns a list of permutations, isn't it so we return null else for i equal to 0 till less than i plus plus we use that function delete list we delete the i'th element, alright. And then we call it on new with n minus 1, r minus 1 as we discussed previously and we get result and in RES which I have not declared but you have to declare it again in a proper structure and we still do not know how to declare this. We have to append in that one the element L [i] and we have to append in each and every element of RES, the element L [i] in the front of it and we will get in final and we will return this final permutations added. Is that okay?

(Refer Slide Time: 34:06)



So let's work out a quick example then we will see how this program, we will just quickly revise how this program works. Suppose it is {a, b, c}, 3 and 2 then first we will start with 0 and we will call it with {b, c}, 2, 1 right and since it is recursion we will go this way first, we will go down like this and we will call it with c, 2, 0 but we are returning anyway. So when we are left with the single element we will return this element okay. If you are, when r is equal to 0 we actually return l, whatever is l is returned. So here it returns null. Isn't it? What does it return? Here it should return a null list, okay no, no we will continue as we were doing. It returns a null list, that's fine. So this will return a null list then this one will call here and this one will call this and this will return. So after this returns a null list, what will happen to the null list?

Let us go back to the program. In the loop you delete an element, you have got a null list returned and you append in that null list L[i]. You append in the null list L[i], here what was L[i] when we called it b? It was called with 0, here it will be called with c, b will be called this side, c will be called this side. So once this null is returned this is recomposed by the append function to give you a tuple b, right.

Now this side it will delete c and it will call it with {b}, 2, 0 sorry 1, 0, this will be 1, 0 this will be 1, 0. This will again return a null, this will append c to this null to prepare c and here it is not actually returned out final, it is whatever you get here you will append it to final in the loop because from this side if you get some result and from the left side and right side say from the left side you get something b, from the right side you get and recompose and get c, you will have to make a union of the two. So in the loop we can make this union and you initialize final in the beginning of the program before here to be null and then you make a union of it. So this will return a list with b and c and this was called with a, so this will form the res... what will be formed after the append? You will get a b and a c.

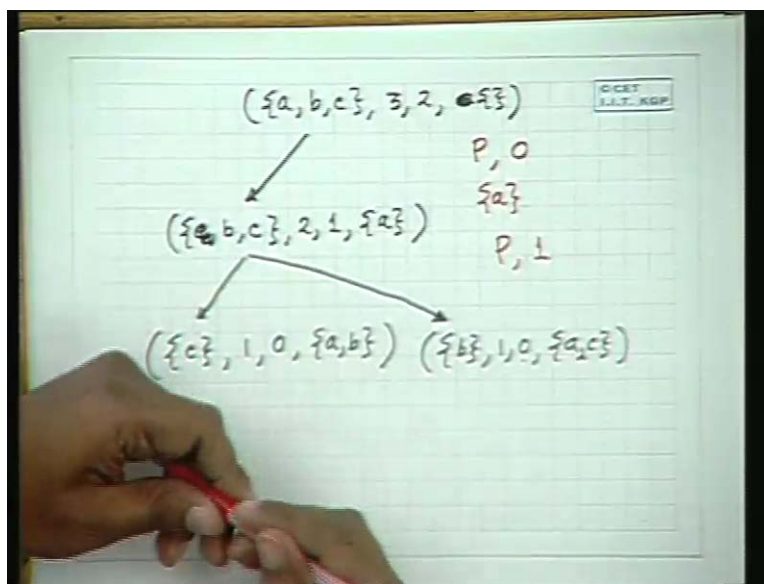
Similarly, you call it with b and similarly you call it with c and you get the three results this way and you combine it. This was the idea which we had. So this is the program in some sense but still we have not been able to declare, we have been able to declare this array as a character array and a string. We still are in trouble in declaring this data structure final and we have to have some means of declaring arrays of strings. Isn't it? You can use this and declare arrays of strings and do it but we will not do that immediately, we will see another simple idea of programming and recursion and solve the problem in a much more simpler manner.

Now let us have a look at this point. At this point we have got this a and we have got b and once we reach 0, this part corresponds to a and b. When you compose, when you go back, null will be appended to b and a will be appended in front of b. If you have a look at how the recursion proceeds, a b will come from this, from this side a c will come and if you go more in more details from this side b a will come, from this side b c will come, from this side c a will come and from this side c b will come. So instead of declaring out what we want, suppose we only want to print the result. Then at this point we could have printed the result provided what, can anybody tell me. Provided what? Provided at this point, see this point we have lost that we have come through a and b.

So we don't know that we have come through a and b but if we have this information we could have printed it here. Here we could have printed a and c, here we could have printed b and a. can anybody suggest how we will do that? So just you pass it through the parameters that is when you are in, when you are calling a b c with 3 2 you have, you call it with null that is at this point up to here nothing has come and then when you come here with b c 2 1, what do you do? You store a.

So let us see if in slightly more details then we will get a better picture. a b c that is we can pass our computation instead of always recomposing it back, we can pass our computation down through the parameters also, this was null.

(Refer Slide Time: 36:28)



And you just pass it down **a sorry**, this was b c 2 1 a, here c 1 0 a b and once this is 0 at this point we just print. Next time it will come out here, print and return we don't have to return any value. Then again it will come here, it will be b 1 0 and a c. And once r is 0, you just print it. So this way you can print a b come here print a b, come here print, come here and so on, alright. So now the program will look like this. What we will do is like we have used the string list character array l for the list, n and r we will use another character array p for storing, for passing down the strings and n will be indicating at what point you have to insert it here that is you will initialize n to 0 and call it.

(Refer Slide Time: 37:27)

```
permut(L, n, r, P, m)
char L[], P[];
int n, r, m
{ if (r==0) { P[m]='0';
              print(P); return; }
  else {
    for (i=0; i<n; i++) {
      P[m]=L[i];
      del_list(L, i, n, NEW);
      permut(NEW, n-1, r-1, P, m+1);
    }
    return;
  }
}
```

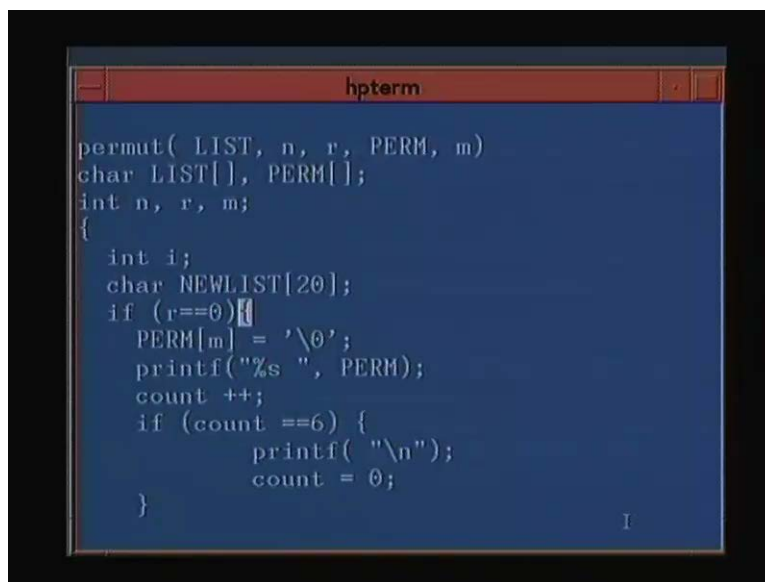
To indicate that this is null, we will pass p, 0. If this is a, if p is a we will pass p with 1 saying that if you want to insert another element, you insert it at the next position. So the program goes like this permut L, n, r, p, m these are two character strings, n, r, m are integers. If r is equal to 0, p [n] is made back slash 0 to end the string and print p in % s format and return, alright. Otherwise again for i equal to 0 to n, you have to find out what is L [i] and you have to store it in p [n] because we are going to pass this down, again delete list and get the new list and call new same n minus 1, same r minus 1, p and this gets incremented by 1. Is that okay?

(Refer Slide Time: 38:00)

```
hpterm
#include<stdio.h>
int count=0;
main()
{
  char LIST[20], PERM[20];
  int n, r;
  printf("Give n and r:");
  scanf("%d%d", &n, &r);
  printf("\n Give the characters as a string
: \n");
  scanf("%s", LIST);
  printf("\n The list is %s \n", LIST);
  permut(LIST, n, r, PERM,0);
  printf("\n");
}
```

So if we just do this we will be able to solve our permutation problem using simple character strings and arrays. We would also like to do it using the data structures, list and other things which we shall come through, any questions. So let's have a look at the program now. Let's remove the comments. So this is the main program which we were working on character array list, integer, give n and r, read n and r, give the characters as a string, print the string and then call it with list, n, r, perm or which is p initialized to 0. We have already seen delete list and we know how it functions, so we are left to see permutations. This is identical to what we wrote down so n, r, perm and n these are two character arrays, n, r, m are integers and we declare new list that is once we delete an element from the list, we will store it in new list so that is why we require this new list.

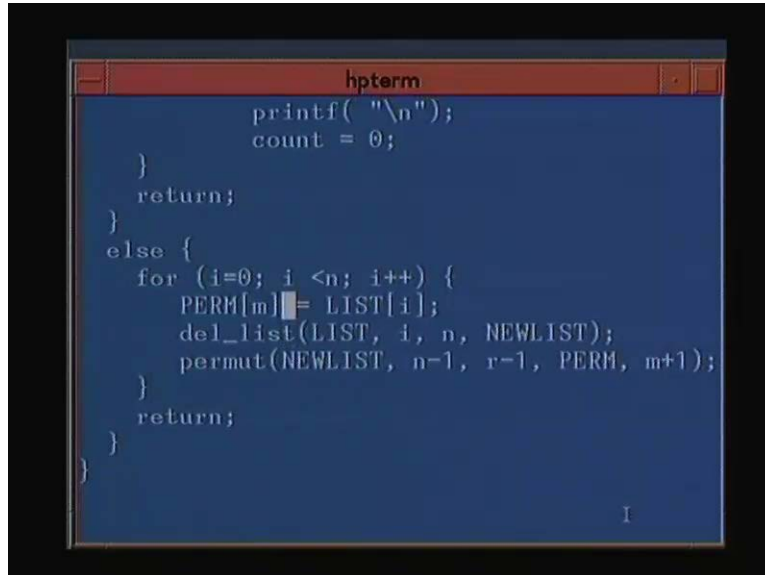
(Refer Slide Time: 38:45)

A screenshot of a terminal window titled 'hpterm' with a dark blue background and white text. The code shown is a C function named 'permut' that takes parameters LIST, n, r, PERM, and m. It declares a character array 'NEWLIST' of size 20. The function logic includes: if (r==0) PERM[m] = '\\0'; printf("%s ", PERM); count++; if (count ==6) { printf("\\n"); count = 0; }. The code is enclosed in curly braces. A small 'I' cursor is visible at the bottom right of the code area.

```
permut( LIST, n, r, PERM, m)
char LIST[], PERM[];
int n, r, m;
{
    int i;
    char NEWLIST[20];
    if (r==0){
        PERM[m] = '\\0';
        printf("%s ", PERM);
        count ++;
        if (count ==6) {
            printf( "\\n");
            count = 0;
        }
    }
```

If r is equal to 0, perm m we put a backslash zero at n and we print it. And this part is just for pretty printing so that only 6 elements are printed per a line that is why I have just, if count is 6, I put a backslash n. This is just for a pretty printing there is nothing else here. So we put do perm m equal to backslash zero and do a printf else and return obviously and else this is this part.

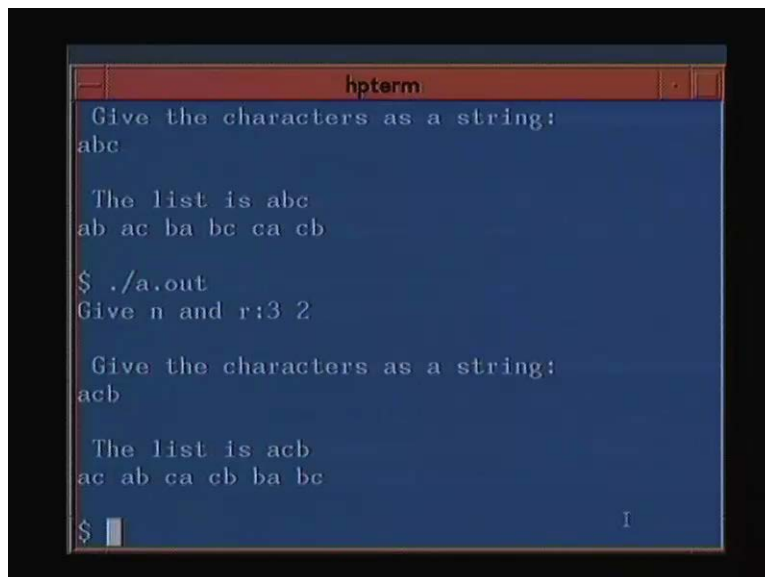
(Refer Slide Time: 39:18)



```
hpterm
    printf( "\n");
    count = 0;
}
return;
}
else {
    for (i=0; i <n; i++) {
        PERM[m] = LIST[i];
        del_list(LIST, i, n, NEWLIST);
        permut(NEWLIST, n-1, r-1, PERM, m+1);
    }
    return;
}
}
```

For i equal to 0 till i less than n i plus plus, perm m is equal to list i, delete list, i, n to the new list and call it with new list, n minus 1, r minus 1, perm and m plus 1. So this is what we were writing. So let's give 3 and 2 a b c, we get a b a c b a b c and so on c a and a b. again you give 3 and 2 and if you give a c b, you will get it in another order because depending on the way the depth first recursion goes.

(Refer Slide Time: 40:13)



```
hpterm
Give the characters as a string:
abc

The list is abc
ab ac ba bc ca cb

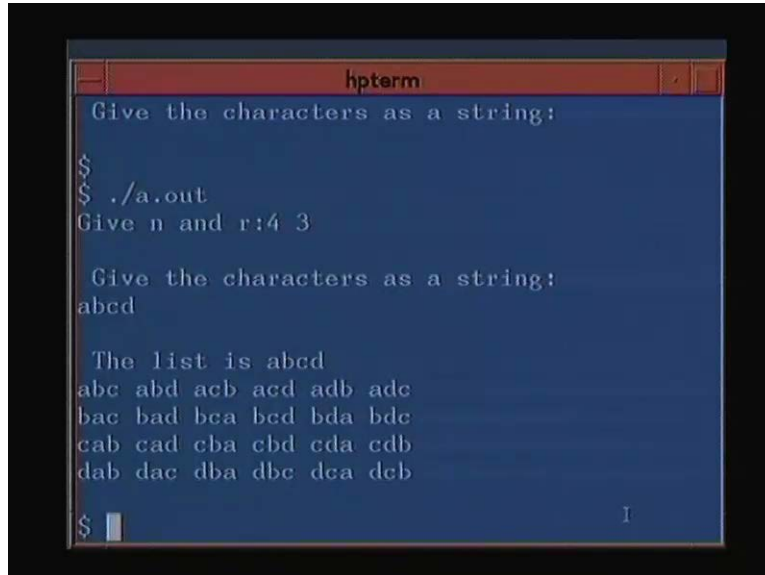
$ ./a.out
Give n and r:3 2

Give the characters as a string:
acb

The list is acb
ac ab ca cb ba bc

$
```

(Refer Slide Time: 40:43)



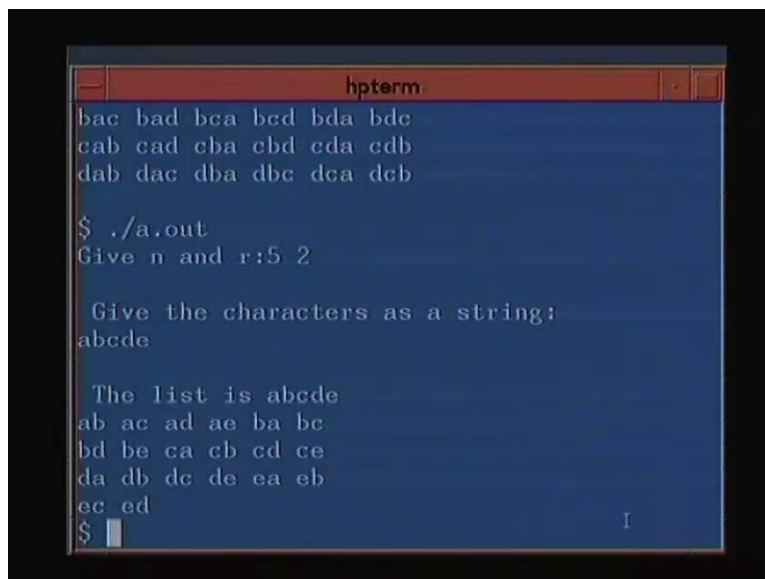
```
hpterm
Give the characters as a string:
$
$ ./a.out
Give n and r:4 3

Give the characters as a string:
abcd

The list is abcd
abc abd acb acd adb adc
bac bad bca bcd bda bdc
cab cad cba cbd cda cdb
dab dac dba dbc dca dcb
$
```

It is dependent on the way in which the depth first recursion goes, so a c a b, am sorry. So with 4 and 3 and 4 characters, you get 24 such outputs, all the possible 24 outputs you can get. And if you want to see, so this way you can give various inputs. If you give a b c d e and you will get 20 such outputs.

(Refer Slide Time: 40:54)



```
hpterm
bac bad bca bcd bda bdc
cab cad cba cbd cda cdb
dab dac dba dbc dca dcb
$ ./a.out
Give n and r:5 2

Give the characters as a string:
abcde

The list is abcde
ab ac ad ae ba bc
bd be ca cb cd ce
da db dc de ea eb
ec ed
$
```

And if you want to see exactly how the program works, it is very important to just print out the value of perm, print out the list perm here at every instance and you will know exactly what are the steps of recursion. In order to understand, suppose you have written this program and you want to understand how the program works and what you do is as

soon as you enter this, you print out the value of the parameter. All the parameters you just print out here and you will know which program is entered and then so you will know the calling sequence and that will give you a perfectly good idea about exactly how the program is working.

So in any recursive program in order to understand the depth first nature of the program, it is better to put in some statements to watch or debug and just find out exactly the sequence. If you have any doubt, just use the computer write a program and just check the sequence. So I hope this program is okay and we have done some simple exercises on characters and strings, I would expect you to read up some more examples on the string functions specially the string length, string compare, string copy and just use them and implement them to find out how strings are used. Also have a look at the functions getchar which obtains a single character, alright. We will see an example of getchar in our subsequent example when we talk of something else. So, we will stop here for this class.