

Problem Solving through Programming in C
Prof. Anupam Basu
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Lecture – 38
Scanf and Printf Functions; Function Prototype

In the earlier lectures, we have looked at functions and we will still be looking at functions. We have also looked at the library functions and just to recapitulate that, whenever we are using some library functions in our program, then we have to some mathematical library functions, then we have to hash include math dot h. And while we link it, we have while we compile it, we will also have to link it with the maths library with the option minus l m as we have seen.

Now, today, since now, we have got an idea of what functions are. Let us have a relook at our old friend's scanf and printf, which we were using a number of times. Now, scanf and printf are nothing but functions. These are also some library functions and those are included whenever we use include s t d I o dot h. Now, you see scanf and printf being functions when we enter data, let us look at this structure.

(Refer Slide Time: 01:29)

Entering input data :: scanf function

- General syntax:
`scanf (control string, arg1, arg2, ..., argn);`

The slide includes a handwritten diagram in blue ink. It shows the word funcn followed by (, "%.d", ., x,). A blue arrow points from the funcn part to the `scanf` in the general syntax above. Below the x is a vertical ellipsis, and below that is scanf followed by (and ... and).

23

We write scanf followed by some parameters. We have also seen that, a function is a function is nothing but, something like f; say, function name followed by some parameters. Here, this scanf is that function name and the parameters are the control

string; that means, say for scanf you have got something like percentage d and then x alright. So, this is the control string and these are the arguments alright. So, both of these are arguments to the functions scanf.

So, scanf is nothing but a function. So, whenever in my program, suppose I am writing a program and here I use scanf with some parameters. Like this, then it is nothing but a call to a system function. A system function which actually does the task of reading the data from the input input device.

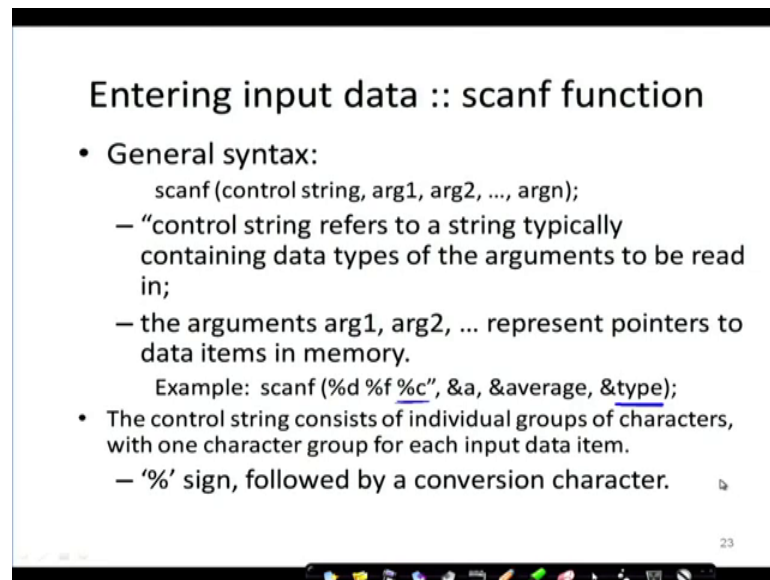
(Refer Slide Time: 02:56)

The slide is titled "Entering input data :: scanf function". It contains a bullet point for "General syntax:" followed by the code `scanf (control string, arg1, arg2, ..., argn);`. Below this, there are two explanatory lines: "– 'control string refers to a string typically containing data types of the arguments to be read in;" and "– the arguments arg1, arg2, ... represent pointers to data items in memory." To the left of the code, there is a hand-drawn diagram of a memory cell represented as a rectangle with a horizontal line through its center. An arrow points from the left side of the rectangle to the letter 'a' next to it. To the right of the diagram, there is a handwritten code snippet: `scanf ("%d", &a)`. In this snippet, an arrow points from the `&` symbol to the letter 'a', which is underlined.

So, a control string refers to typically the data types of the arguments. We have seen percentage d percentage f etcetera. And the arguments are pointers to data items in memory. These arguments in scanf are what are; the typical arguments in scanf? They are say, whenever I use scanf, say percentage d then and a or something like that right?

Now, this and is nothing but an address of the variable a. So, in my memory, wherever the variable a is, the variable a the compiler is allocated this memory location, for the variable a and I am passing the address of that so; that means, it is a pointer pointing to some address or a pointer that is pointing to this particular location in the case of scanf, ok. So, that is why, it said the arguments these arguments arg 1 arg 2 arg n are representing nothing but pointers to data items in the memory .

(Refer Slide Time: 04:21)



Entering input data :: scanf function

- General syntax:

```
scanf (control string, arg1, arg2, ..., argn);
```

 - “control string refers to a string typically containing data types of the arguments to be read in;
 - the arguments arg1, arg2, ... represent pointers to data items in memory.

Example: `scanf ("%d %f %c", &a, &average, &type);`
- The control string consists of individual groups of characters, with one character group for each input data item.
 - ‘%’ sign, followed by a conversion character.

23

So, the example, a typical example is, say, percentage d percentage f percentage c to which is nothing but the control string, it is designating that and a; a is an integer and a is a pointer to a. And average is a floating-point number. And average is a pointer to average. And type is a character type variable. And type is, why is type a character type variable? Because, I have put in here and c since I have put in here and c that tells me that this type is a character type variable and type is a pointer to the variable type.

The control string consists of individual groups of characters with one-character group for each input data item. So, this is one-character group for this a type. So, percentage we have seen that percentage sign means, is a conversion character.

(Refer Slide Time: 05:29).

– Commonly used conversion characters:

c	single character	←
d	decimal integer	←
f	floating-point number	←
s	string terminated by null character	←
X	hexadecimal integer	

0...9
s
X

24

The commonly used conversion characters we have already seen some of them, these are known to us, percentage d is for decimal or integers, percentage f for floating point numbers, percentage c for single character, percentage s is a string. Whenever, I am reading something variable as a string, we have discussed that in that case, we are reading the string using percentage s and that is a string is always terminated by null character.

Similarly, percentage x denotes that, the number that I am reading is a hexadecimal character. Hexadecimal means decimal is with a base ten; that means, 0 to 9 are my elements. So, all the numbers I have, I am describing using 0 to 9. That is my alphabet set. Whereas, in the case of hexadecimal, the base is 16 and so, base being 16, I have got 0 1 2 3 4 5 6 7 8 9. These 10 numbers followed by A for 10, B for 11, C 12, D 13, E 14 and F 15.

(Refer Slide Time: 06:48)

– Commonly used conversion characters:

- c single character
- d decimal integer
- f floating-point number
- s string terminated by null character
- X hexadecimal integer

0 1 2 3 4 5 6 7 8 9 }
A B C D E F }
 11×16^0
 $+ 10 \times 16^1$
 $= 160 + 11 = 171$
 $16^2 \ 16^1 \ 16^0$
 $2^2 \ 2^1 \ 2^0$
□ □ □

So, up to that, 0 to 15, I can have. So, in a 16, when I work with a base 16 and that is known as hexadecimal. Now so, so you can just establishing a similarity, analogy with our binary numbers system that we have seen, we can have, say, a number A B.

Now, A B is a string. So, the position string position weights are, first position is 16 to the power 0, second position is 16 to the power 1, third position is 16 to the power 2, like that. In the case of binary, we had the first position way, it was 2 to the power 0, 2 to the power 1, 2 to the power 2, like that right. Now and each of these positions can be filled up in the case of binary by 0 and 1, but here, in the case of hexadecimal, it can be filled up with any of these 0 to a 0 to F.

So, A B, when I say; that means, B is what in decimal B is 11 12. So, 12 times what is the weight of this position 16 to the power 0 plus A is 11 sorry, sorry, sorry, sorry, I am, I am, I am sorry. this is 10. 10 to sorry let me correct that, A is 10 and B is 11. So, it is 11 into 16 to the power 0 and 10 times 16 to the power 1. So, how much is that coming to? 160 plus 11. So, that is the number 171. Now, if I had to represent 171 in binary, I would have required much more number of bits right.

And since it is hexadecimal, another so, here we using hexadecimal, I am being able to do that using 2 hexadecimal bits, which we can call hex x, but, let us see A B.

(Refer Slide Time: 09:54)

– Commonly used conversion characters:

- c single character
- d decimal integer
- f floating-point number
- s string terminated by null character
- X hexadecimal integer

A	B
1010	1011

171

24

A being 10 and B being 11, in binary what would that be B is 11; that means, 8 2 1, this is 11 and 10 is 8 2 that is all. So, I would have required 4 bits for representing 171, which I am representing using hex x alright. So, that is a hexadecimal integer. So, if I if I print something in hex in this in this format, then it would be printed like 171 will be printed as A B. Otherwise, it will be printed as in the binary number or in the decimal number. If I do it with percentage d will be printed like this.

So, it was a brief introduction to hexadecimal numbers, but that you can read up yourself. So, let us proceed.

(Refer Slide Time: 11:03)

– Commonly used conversion characters:

- c single character
- d decimal integer
- f floating-point number
- s string terminated by null character
- X hexadecimal integer

– We can also specify the maximum field-width of a data item, by specifying a number indicating the field width before the conversion character.

Example: `scanf(\"%3d %5d\", &a, &b);`

We can also specify the maximum field width of a data item. Now, this is something I had purposefully evaded till now, just to not to make the things complicated. I want that you first get a custom to the normal features of c. But now, we are in a now, I think you have you are in a position where you can write programs. Here, I am introducing the notion of field width. A data will be printed within a width, right? So many positions will be given to that number. data item the number indicating the field width before the conversion character.

For example, let us make it clear. For example, percentage 3 d, percentage 5 d for A and B; that means, A will be expected to be of sorry; A will be expected to be of 3 positions; say, percentage it is 3 d decimal 171 whereas, B will be 5 d. So, there will be 5 positions for this 1 2 3 4 5. So, may be 52732. Now, if I want to print 171 or read 171 using the format 5 d, then what should I print? I should print a sorry I should provide the data for the 5 fields. I should put it blank, blank or 0 0 then 171 assuming, it is right justify alright.

So, this is the width. This is the width that we are talking about. We can specify how many what is the when I am reading what is the format of the data? How many widths, how many positions it is taking for as the data, right?

(Refer Slide Time: 13:32)

171 5 3 2 7

- Commonly used conversion characters:
 - c single character
 - d decimal integer
 - f floating-point number
 - s string terminated by null character
 - X hexadecimal integer
- We can also specify the maximum field-width of a data item, by specifying a number indicating the field width before the conversion character.
 - Example: `scanf ("%3d %5d", &a, &b);`

24

So, say for example, if I have.171 5 3 2 then 7, then, if I read it, the first data if field is feed is this, I type to the keyboard and if it is in percentage 3 d it will be fine ok. So, I will read that.

(Refer Slide Time: 13:56)

Writing output data :: printf function

- General syntax:

```
printf (control string, arg1, arg2, ..., argn);
```

 - “control string refers to a string containing formatting information and data types of the arguments to be output;
 - the arguments arg1, arg2, ... represent the individual output data items.
- The conversion characters are the same as in scanf.

25

Again, now coming to writing now, this printf is also a system function. Just like scanf printf is also a system function and here also, since you know already, how do you write printf? I write printf using a control string within the double code and then followed by

this number of arguments. So, the same thing will hold here. Control string refers to the string containing formatting information and data types of the arguments to be output .

So, now, here while printing the arg 1 arg 2, these are the representing the individual data items. In the case of scanf, they were pointers to the data items. Here, they are individual data items that I have told you earlier. That is why, we use ampersand a ampersand b in the case of scanf. But in the case of printf, we just write a b, ok. Because, this a b are directly referring to the variables the data items. The conversion characters are the same as scanf.

(Refer Slide Time: 15:16)

• Examples:

```
printf ("The average of %d and %d is %f", a, b, avg);  
printf ("Hello \nGood \nMorning \n");  
printf ("%3d %3d %5d", a, b, a*b+2);
```

The diagram shows a horizontal array of six boxes containing the digits 1, 7, 1, 6, 3, and 2. Below this array, a smaller box contains the digits 1, 7, and 12, illustrating how the output is formatted according to the printf format string.

So, let us look at some examples. Printf the average of a and b is avg and that avg is percentage f, you can see that right. So, avg is a float.

Now, I could have also done this that, so, this is nothing but, there is a control string , here you see, I have said that 3 d 3 d 5 d. So, a will be printed. So, when do you printed a will be printed, there is no new line here you can see.

So, a b and a times b plus 2 will be printed side by side. So, a will be printed on 3 fields like, say 171, b b will be printed on 3 next 3 there is no space given. So, this will be b 6 3 2 and the expression. Here a times b plus 2 will have have 5 spaces 5 positions given now; obviously, if my data for result for a was 1712, then only this much will be printed. Then the rest will be left out, ok.

(Refer Slide Time: 17:02)

- Examples:

```
printf ("The average of %d and %d is %f", a, b, avg);
printf ("Hello \nGood \nMorning \n");
printf ("%3d %3d %5d", a, b, a*b+2);
printf ("%7.2f %5.1f", x, y);
```

101.52
0101.5

26

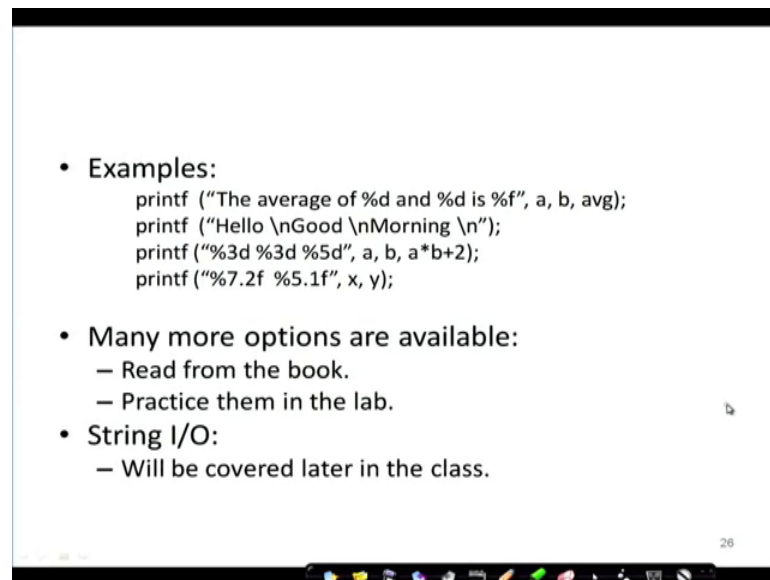
So, here you see for the percentage f.

Now, for this, I think it is not very difficult to understand. You can read them in the any textbook. It is clearly discussed in most of the textbooks. So, I recommend you to go through these formats and as you practice, you will keep that in mind, but just to tell you this. So, floating point number will have 2 parts, right? One is the integer part and the other is a decimal part. So, suppose here, when I say 7.2 f x is a floating-point number, which is being printed in the formats 7.2 f. So, this entire width is 7 positions and the 2 positions here are kept for the decimal. So, these 2 are kept for the decimal and we imagine, we will have a decimal point here and the rest 5 will be for the integer part.

For y what will happen? For y the total field is 5, out of which, so total field is 5 out of which only 1 position is kept after the fraction. So, here, I assume the this point.

And so, there are 4 positions to represent an integer and one position to deserve represent a fraction. So, if my result was 101.52 in this format, if I print that, then I will get 0101 or this 0 can be may not be printed.

(Refer Slide Time: 19:09)



• Examples:

```
printf ("The average of %d and %d is %f", a, b, avg);
printf ("Hello \nGood \nMorning \n");
printf ("%3d %3d %5d", a, b, a*b+2);
printf ("%7.2f %5.1f", x, y);
```

• Many more options are available:

- Read from the book.
- Practice them in the lab.

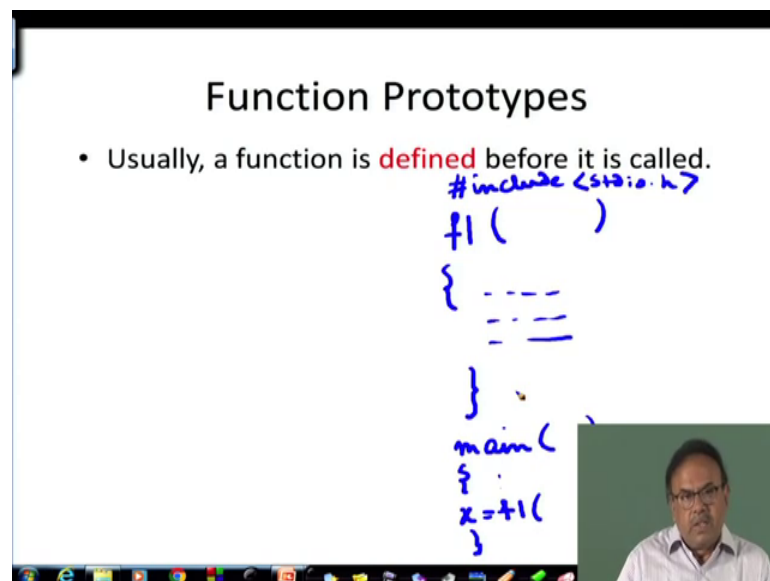
• String I/O:

- Will be covered later in the class.

26

Many more options are available. You can read from the book and we will do this later.


(Refer Slide Time: 19:13)



Function Prototypes

- Usually, a function is **defined** before it is called.

```
#include <stdio.h>
f1 ( )
{
  ---
  ---
}
main ( )
{
  x = f1 ( )
}
```



Now, coming to a very important concept ah, that is, function prototypes. This is possibly new name that you are getting. What are these prototypes? Now, we are going to write functions and those functions, one of the functions which must be there is the main function, right? Typically, people write first the main function and then the first the functions and then the main function, but you are free not to do that also. So, usually a function is defined before it is called. So, typically, what we do is, when our main our

programs starts, so, we start the program starts from here, we define of we write a function with some parameters here and the body of the function is here, then, we write, say, main. And inside main, I call f 1 this assigned to some variable x like that. So, this is the typically some what we do alright. After I did, this include s t d I o dot h. I started with the function first; that is, one option now, but always, that is not done.

(Refer Slide Time: 20:58)

Function Prototypes

- Usually, a function is **defined** before it is called.
 - main() is the last function in the program.
 - Easy for the compiler to identify function definitions in a single scan through the file.

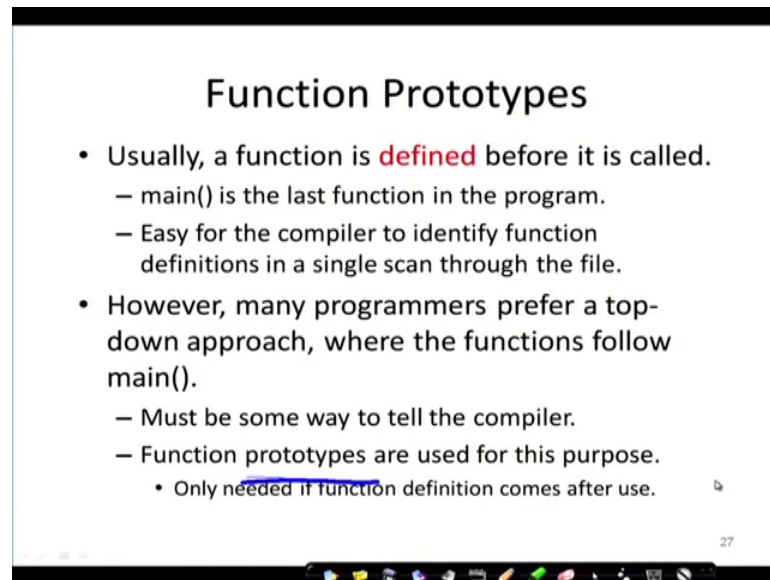
27

So, main in that case, if I define the functions first the functions, that I am going to use, I have thought about that I have designed them. So, first I write those functions and after that, I write the main function. In that case, the main is the last function in the program. So, in that case, the compiler, so, I have got my program here, entire program here. And I have defined the function f 1 here. I have defined function f 2 here. And when I write main here and in main, I refer to f 1 f 2 whichever whenever is this required.

Now, let us look at the compiler think of yourself to be the compiler. Now, if you had started compiling the main first and you would have come to f and f 2, then you would wonder what is that f 1 and f 2. No variables are defined right? You would have taken just like, if you had written x in the main, some function f 1 some function like f 1 a b right. Then, this f 1 is not recognised; I mean you do not know about that here, but if you write it before, the compiler will actually start looking from the beginning of the page, beginning of the program. You will understand, a function has been defined here, I understand that function. So, I know that and the I know what this function does it

compiles that and f 2 also. So, when it comes to in this reduction, when it comes to the main and finds f 1 and f 2, it already knows that. So, there is no problem. So, easy for the compiler to identify the functions, when it scans through the file this a file right, the whole thing is a file

(Refer Slide Time: 23:22)



Function Prototypes

- Usually, a function is **defined** before it is called.
 - main() is the last function in the program.
 - Easy for the compiler to identify function definitions in a single scan through the file.
- However, many programmers prefer a top-down approach, where the functions follow main().
 - Must be some way to tell the compiler.
 - Function prototypes are used for this purpose.
 - Only needed if function definition comes after use.

27

However, always that is not done many programmers prefer a top down approach, where I will first define main and then I will put the functions, but then what will happen to the compiler? The compiler if it starts with main, it is starting from the beginning and if it encounters f 1 or f 2 in the body of the main function, then it will get confused what is it just like, if I had got a variable x being used and it is not defined there been error because, I do not know what this x is, what type it is right? So, I have to declare that before there. Similarly, for functions there must be some way to tell the compiler that is the function and for that function prototypes are used function prototypes are used.

(Refer Slide Time: 24:32)

Function Prototype (Contd.)

- Function prototypes are usually written at the beginning of a program, ahead of any functions (including main()).
- Examples:

```
int gcd (int A, int B);  
void div7 (int number);
```

The diagram shows a square box with a wavy line inside, representing a function. Two arrows labeled 'A' and 'B' point into the top of the box. An arrow points from the bottom of the box to a circle containing the word 'int', representing the return type. The box is labeled 'gcd' on the right side. A blue arrow points from the first example code line to the box.

28

Let us look at this needed we the function comes later. So, function prototypes let us look at . Are usually written at the beginning of the program ahead of any other functions including main, say for example, this is a prototype. This is a prototype, `int gcd (int a, int b)` ok. So, a and b are the parameters and gcd is a function of return type int. So, I know just by this prototype by this, I know the basic whatever is a body whatever is a body of gcd, I know it is interface, I know it is input output. What is it is input output? There will be some a coming some parameter.

There are 2 parameters internally, they are named as a and b. I also know that, this is an integer and this is also is integer and I know that it is name is gcd and I also know that, it is output is an integer. Whatever is in between, I am not bothered about right now this much I know. So, I will expect whenever gcd will come, I will allocate 2 integers space for that and one return facility should be there.

Similarly, let us look at the second one `div 7 int number`.

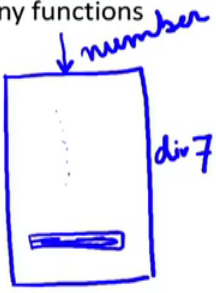
(Refer Slide Time: 26:24)

Function Prototype (Contd.)

- Function prototypes are usually written at the beginning of a program, ahead of any functions (including main()).
- Examples:

```
int gcd (int A, int B);  
void div7 (int number);
```

if (number % 7 == 0)



The diagram shows a rectangular box representing a function. An arrow labeled 'number' points into the top of the box. The box is labeled 'div7' on the right side. Inside the box, there is a dashed vertical line and a horizontal line at the bottom, representing the function's body.

28

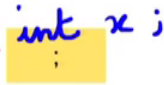
So, the prototype tells that, it is a function. Its name is div 7. You can guess div 7 means, is testing the divisibility by 7 and there is one integer that is being fed and that integer's name is number and void is a type; that means, it is not returning anything. Maybe, it is printing something from here, it just checks, takes a number; it sees whether it is divisible by 7 and prints it here divisible by 7, not divisible by 7, whatever you know. How to find divisibility by 7? So, you take the number and find the mod of that with 7 and if this is equal to 0, then divisible by 7, otherwise not. So, that is in the body of the function and here, we just do the printing. So, I am not returning anything to the calling function. That is why, this is known as void. Void is a valid type.

(Refer Slide Time: 27:43)

Function Prototype (Contd.)

- Function prototypes are usually written at the beginning of a program, ahead of any functions (including main()).
- Examples:

```
int gcd (int A, int B);  
void div7 (int number);
```



- Note the semicolon at the end of the line.
- The argument names can be different; but it is a good practice to use the same names as in the function definition.

28

Note the semicolon now here this very important. Whenever for hash include the s t d I o dot h, I did not give a semicolon. But whenever I am declaring a function of function prototype, I have to give a semicolon just as I had given a semicolon when I declare something like int x semicolon.

Similarly, here, this semicolon is very important. It just like a declaration. So, the argument names can be different, but it is I mean it is a good practice to use the same names in the function. Now ah, so, what is being said here is that, although I am using here, showing that A and A, but when I am actually writing the function later say, sorry.

(Refer Slide Time: 28:46)

Function Prototype (Contd.)

- Function prototypes are usually written at the beginning of a program, ahead of any functions (including main()).
- Examples:

```
int gcd (int A, int B);  
void div7 (int number);
```

int gcd (int X, int Y);

- Note the semicolon at the end of the line.
- The argument names can be different; but it is a good practice to use the same names as in the function definition.

28

When I am writing the function later, when I wrote later on int g c d, I can make, let us space, sorry I can write int g c d int x comma int y that is also possible ok. Because, my prototype just said that, 2 integer places. Now actually, I can change the name, change the name of the place, but that is not that important. I mean that is not I mean, usually it is better if we can keep both of them.

(Refer Slide Time: 29:33)

Function Prototype: Examples

```
#include <stdio.h>  
int ncr (int n, int r);  
int fact (int n);  
  
main()  
{  
    int i, m, n, sum=0;  
    printf("Input m and n \n");  
    scanf ("%d %d", &m, &n);  
  
    for (i=1; i<=m; i+=2)  
        sum = sum + ncr (n, i);  
  
    printf ("Result: %d \n", sum);  
}
```

```
int ncr (int n, int r)  
{  
    return (fact(n) / fact(r) / fact(n-r));  
}  
  
int fact (int n)  
{  
    int i, temp=1;  
    for (i=1; i<=n; i++)  
        temp *= i;  
    return (temp);  
}
```

29

So, here is a function prototype example. I will continue with this in the next lecture. I will discuss this and we will move to some other important feature of function like parameter passing in the next lecture onwards.