

Introduction to Programming in C

Department of Computer Science and Engineering

(Refer Slide Time: 00:09)

```
# include <stdio.h>
main()
{
  printf("Welcome to ");
  printf("C Programming");
}
```

sample.c

Tell compiler to include the standard input output library

In this session let see another simple program, and try to study what is going on in there. So, here I have slightly more complicated program, then what we just saw. This has two printf statements; once is welcome to, and the second says C programming. So, it is slightly more sophisticated then the code that we have seen. So, to recap the first line `# include <stdio.h>`, tells the compiler to include the standard output library.

(Refer Slide Time: 00:42)

```
# include <stdio.h>
main()
{
  printf("Welcome to ");
  printf("C Programming");
}
```

sample.c

There are two statements in main
→ Statement 1: Statement 2

- Each statement is terminated by semi-colon;
- Curly braces enclose a set of statements.
- Statements are executed in sequence.

Compile and Run

```
%gcc sample.c
%.a.out
Welcome to C Programming%
```

Defines the main function. The brackets () show that main function takes no arguments.

Execution always begins from the first statement of main function.

First { signals the beginning of the body of main. Last } signals its end.

Then we have the main function; the open and close brackets immediately after name show that main is a function, execution always begins at the first line of the main statement. Then the body of the function the logic of the function is enclosed within two curly braces; the first curly brace signals - beginning of the function, and the last curly brace says that the function is over here.

This particular name function has two statements; earlier we have just one statement. The each statement as I said before is terminated with a semicolon. So, this is the first semicolon, and this is the second semicolon. The first semicolon says that the statement printf welcome to ends that point, and then followed by the second statement. And the second statement is also terminated by semicolon. Every statement in C is terminated with the semicolon. Curly braces enclose a set of statement, and each statement in a sequence is executed in the exact sequence that we wrote in the code. Now, once we edit this in an editor save the file, now compile, and run the file. So, let us call it `sample.c` or you may rename it any file you want. And then once the compilation is successful, you can run it using `./a.out`, and then it will print welcome to C programming, which was essentially the same messages as we seen before.

(Refer Slide Time: 02:36)

The slide titled "Tracing the Execution" illustrates the execution of a C program. On the left, a code block shows the source code with line numbers 1 through 6. The code includes a header file, a main function, and two printf statements. Below the code, the output is shown as "Welcome to C Programming%". To the right, a list of four bullet points explains the execution process: the program counter starts at the first executable statement, line numbers are provided for clarity, the program is run step-by-step, and it terminates gracefully when main returns.

```
Line No.
1 #include <stdio.h>
2 main()
3 {
4     printf("Welcome to ");
5     printf("C Programming");
6 }
```

Output: `Welcome to C Programming%` (After lines 5,6)

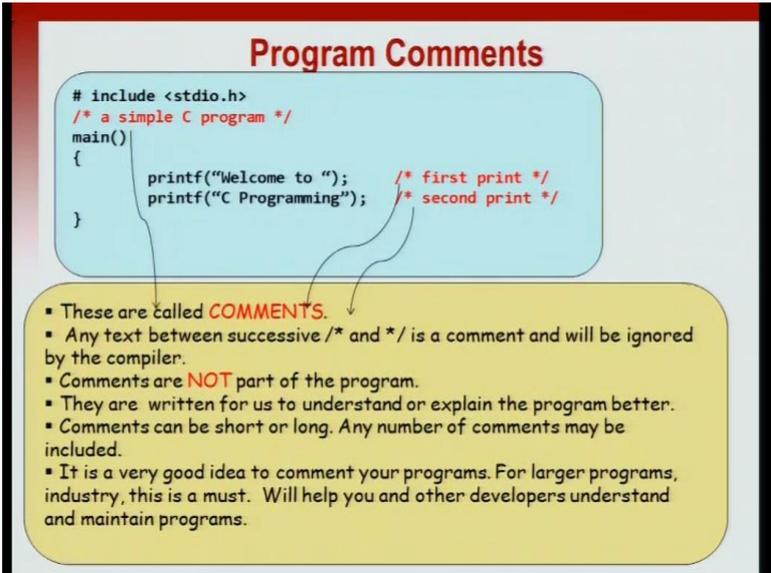
- Program counter starts at the first executable statement of main.
- Line numbers of C program are given for clarity.
- Let us run the program, one step at a time.
- Program terminates gracefully when main ``returns``.

Let us try's what happens when we execute the program. By tracing we mean step by step looking at each statement, and C's see what happens when the program executes. We have what is known as a program counter, which says here is the currently executing

line of program. The program counter starts executing at the first statement of the mean, for is of reference I have given line numbers in the code. Now this is given just for clarity. Now let us just see, what happens when we run the program line by line.

So first we execute the first line of the code, after we are done executing the line 4. So, after we are done executing lines 3 and 4, the message welcome to will be printed on the terminal. This will be followed by the next line, so the next lines is C programming. So, after the next line executes, it will print C programming %. I given this in two different colors to highlight that one was printed by the first line, and otherwise printed in second line, otherwise the colors have nosing, no special meaning. The program terminates when the main finishes execution, and this is what is typically known as returning from the function, we will see this terminology later in the course.

(Refer Slide Time: 04:19)



Program Comments

```
# include <stdio.h>
/* a simple C program */
main()
{
    printf("Welcome to "); /* first print */
    printf("C Programming"); /* second print */
}
```

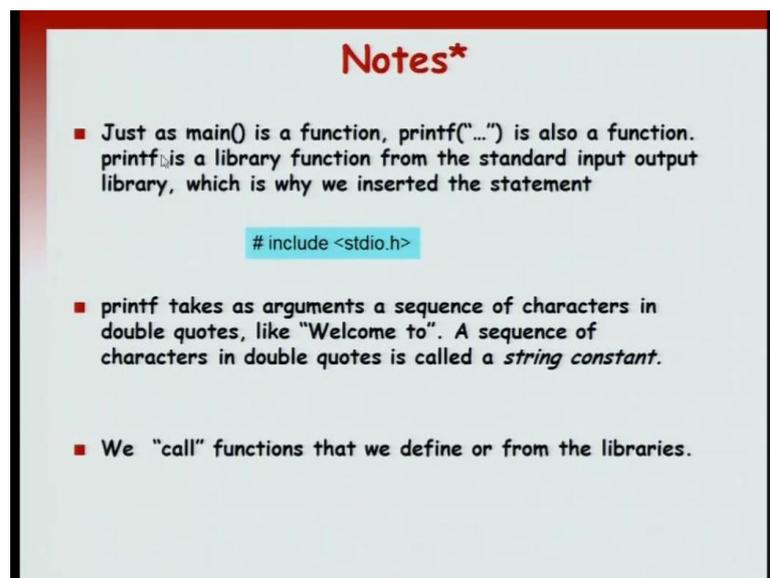
- These are called **COMMENTS**.
- Any text between successive **/*** and ***/** is a comment and will be ignored by the compiler.
- Comments are **NOT** part of the program.
- They are written for us to understand or explain the program better.
- Comments can be short or long. Any number of comments may be included.
- It is a very good idea to comment your programs. For larger programs, industry, this is a must. Will help you and other developers understand and maintain programs.

Now, when you code in addition to the statements which are actually executed, you may also give a few additional remarks; these are what are known as come program comments. For example, the lines a simple C program first print and second print; these are the comments in the code. So, whatever is highlighted in red in the code is what are known as comments. Any text between forward **/***, and then later followed by a ***/**. So, any text between successive **/*** and ***/** is a comment, and it will be ignored by the compiler. So, as far as the compiler is concerned a code with comments is the same as a code without comments. It does not effect the logic of the code. So, comments are not

part of the program; however, it is highly recommend that any program you write, you should comment the code. This is show that other people can understand your code also you yourself looking at the code 4 months later or five months later, it is it may be difficult to understand what you wrote? Much before and comments help you understand the logical of the program.

Now, it is a very good idea to comment your programs, and for lager program it is a must to comment the programs. This is standard industry practice, and even if you participating in large programming project like free software projects, comments are highly encouraged, because it will understand other developers, other programmers to understand your code. So, we will try to follow our own advice most of the programs that we will see in this code, we will comment it, so that it easy to follow the logic of the code.

(Refer Slide Time: 06:15)



Notes*

- Just as `main()` is a function, `printf("...")` is also a function. `printf()` is a library function from the standard input output library, which is why we inserted the statement

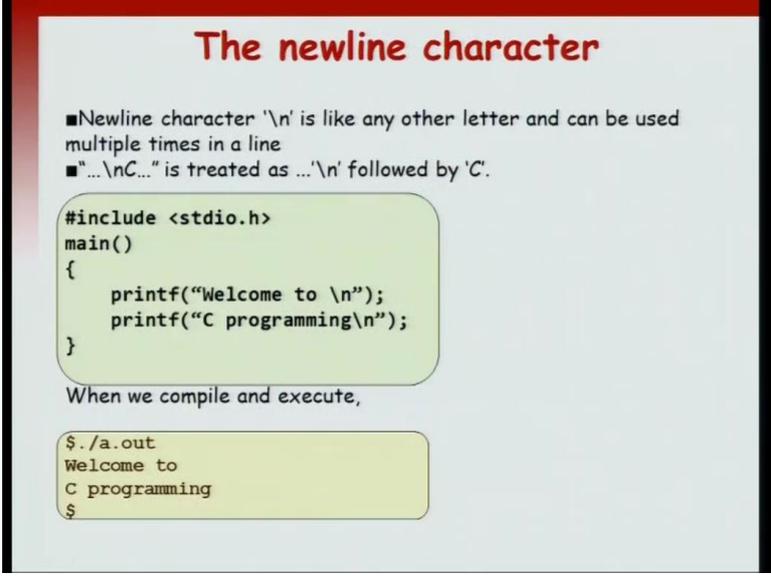
```
#include <stdio.h>
```

- `printf` takes as arguments a sequence of characters in double quotes, like "Welcome to". A sequence of characters in double quotes is called a *string constant*.
- We "call" functions that we define or from the libraries.

Now, a few notes just as a main is a function `printf` is also a function. `printf` is a library function which means that it is given by the C programming language, and we wanted to tell the compiler to include this library function. The statement which set that is this `#include <stdio.h>`. So, `#include <stdio.h>` is the line telling that I want the standard input output library, because that is the library from which I will get the function `printf`. Now what does `printf` do? `printf` takes two arguments, just like arguments to mathematical

So, even though it is single character, it is denoted by two letters. When used in printf it causes the current output line to end, and then printing will start from the new line. So, it is something which says the current line has enter, now whatever you have to print, print it in the next line.

(Refer Slide Time: 09:34)



The newline character

- Newline character '\n' is like any other letter and can be used multiple times in a line
- "... \nC..." is treated as ...'\n' followed by 'C'.

```
#include <stdio.h>
main()
{
    printf("Welcome to \n");
    printf("C programming\n");
}
```

When we compile and execute,

```
$/a.out
Welcome to
C programming
$
```

The new line character `\n` is like any other letter, and can be used multiple times in any particular line. For example, if you have something to print followed by `\n`, followed by C, followed by something to print. Now this will be treated as, so many characters and then a new line followed by C. So, let see a particular example, if you have the old program that we just wrote, but we end each message which a `\n`. So, we have printf welcome to `\n`, printf C programming `\n`. When we compile and execute, we will see something new. So, when we run this `$/a.out`, it will print welcome to, and then the next thing to print is is a `\n` which is a new line. So, printing will start from the next line, and then it will print the next message with C programming. So, it will print that followed by new line. So, the prompt will appear on the line after words. So, new line character is something that is use to make your output messages a little more ((Refer Time: 10:58)) here.

(Refer Slide Time: 11:00)

Last on newlines

■To repeat, newline character '\n' is like any other character. It can be used multiple times. Another example.

```
#include <stdio.h>

main()
{
    printf("Welcome to\n\nC\n");
}
```

■When we compile and execute, we have the following.

```
$/a.out
Welcome to

C
$
```

So, let us just conclude by saying one more thing about new lines, the new line character `\n` is like any other character, and you can use it multiple times even within single message. For example, if I do the same program, but let us say I have welcome to `\n \n C \n`. So, I have repeated occurrences of `\n` in the same message, what it will do is it will print the message welcome to, then it will print a new line, and then it will print another line, and then it will print C followed by new line. So, when you run program you will have welcome to new line, then the blank line, then C, then another line. So, new lines are just like any other characters, the difference is that, because it is a special character, it is represented by two letters. So, it is not represented by a single letter it is represented by `\n`. So, they are together one character in C, call the new line character.