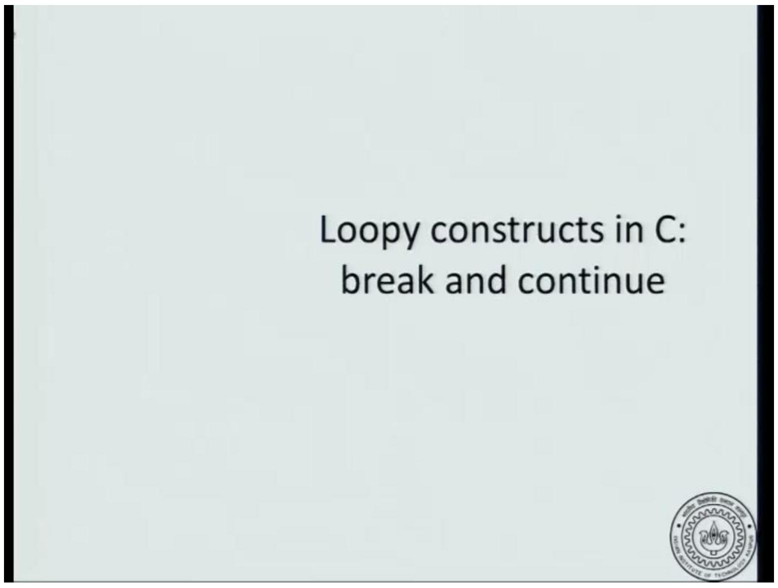
Introduction to Programming in C

Prof. Satyadev Nandakumar
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 17

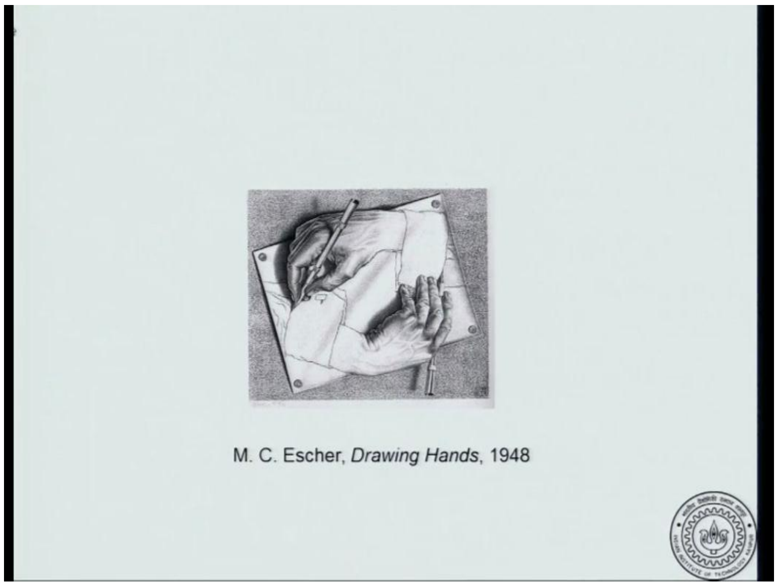
(Refer Slide Time: 00:07)



Loopy constructs in C:
break and continue

In this session we will see one more feature that is present in C associated with loops.

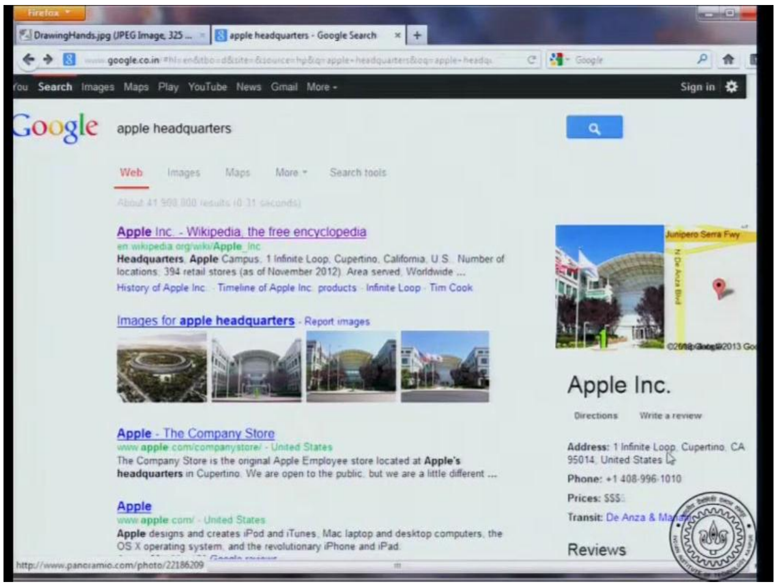
(Refer Slide Time: 00:09)



M. C. Escher, *Drawing Hands*, 1948

So, we will in motivate these statements using the concept of an infinite loop. Here is a drawing that supposed to be a representation of an infinite loop.

(Refer Slide Time: 00:22)


Firefox
DrawingHands.jpg (JPEG Image, 325 ...
apple headquarters - Google Search
google.co.in
You Search Images Maps Play YouTube News Gmail More
Sign in
Google
Web Images Maps More Search tools
Junipero Serra Fwy
N De Anza Blvd

apple headquarters

Apple Inc. - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Apple_Inc
Headquarters. Apple Campus, 1 Infinite Loop, Cupertino, California, U.S.. Number of locations: 394 retail stores (as of November 2012). Area served: Worldwide ...
History of Apple Inc. · Timeline of Apple Inc. products · Infinite Loop · Tim Cook

Images for apple headquarters - Report images

Apple - The Company Store
www.apple.com/companystore/ - United States
The Company Store is the original Apple Employee store located at Apple's headquarters in Cupertino. We are open to the public, but we are a little different ...

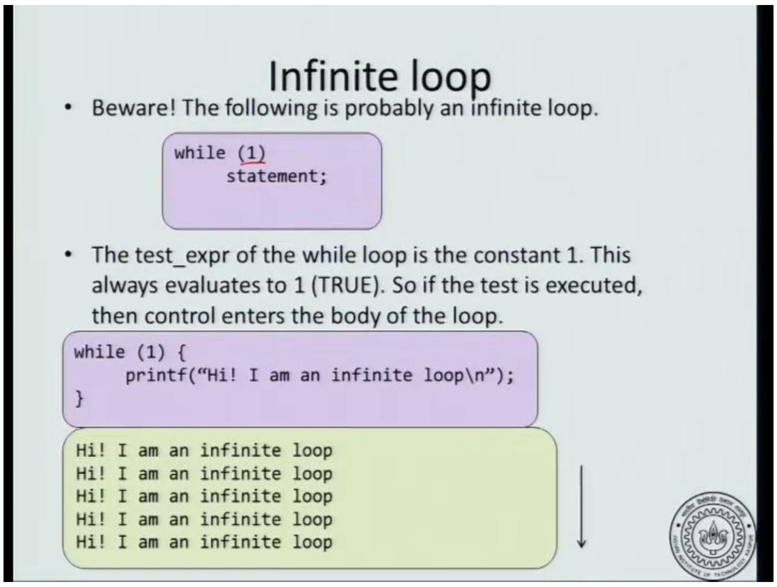
Apple
www.apple.com/ - United States
Apple designs and creates iPod and iTunes, Mac laptop and desktop computers, the OS X operating system, and the revolutionary iPhone and iPad.

Apple Inc.
Directions Write a review
Address: 1 Infinite Loop, Cupertino, CA 95014, United States
Phone: +1 408-996-1010
Prices: $$$
Transit: De Anza & Mariani
Reviews

http://www.panoramio.com/photo/22186209

And a trivia, for example, the apple head quarters; the address is one infinite loop.

(Refer Slide Time: 00:33)



Infinite loop

- Beware! The following is probably an infinite loop.

```
while (1)
      statement;
```

- The test_expr of the while loop is the constant 1. This always evaluates to 1 (TRUE). So if the test is executed, then control enters the body of the loop.

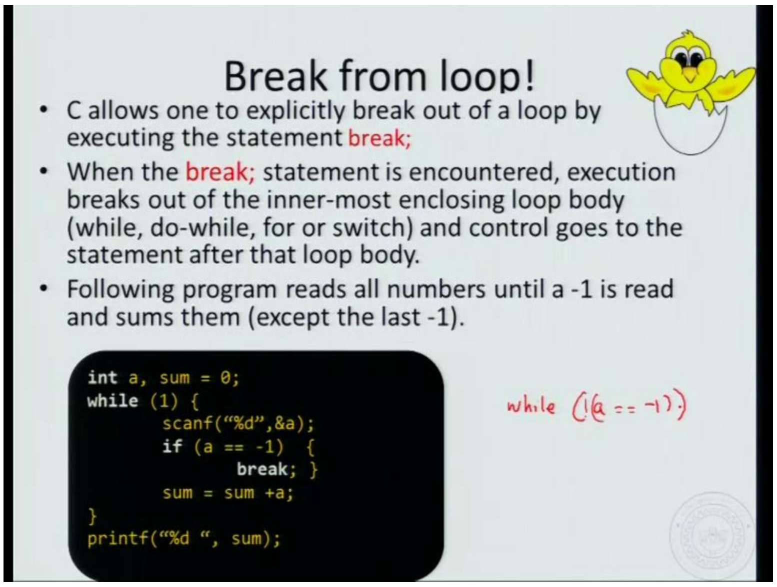
```
while (1) {
     printf("Hi! I am an infinite loop\n");
}
```

```
Hi! I am an infinite loop
Hi! I am an infinite loop
Hi! I am an infinite loop
Hi! I am an infinite loop
Hi! I am an infinite loop
```

So, let us see what is an infinite loop? So, the basic or the simplest kind of infinite loop is when you have a while statement. And the test condition, you can see that it will never be false. So, remember that 1 is true in c. So, this statement means that you will enter the while loop, you will test the condition. The test is true, so, you will execute the statement. You will go back and test the condition again, it is again true, does not changed; it is 1. Therefore, you will enter the statement again. So, you will have an infinity loop.

If the test is executed then the control enters the body of the loop, and this happens without any change. So, let us look at this simple while loop which is while 1. The statement is print f Hi! I am an infinite loop. So, if you will run this code, compile and run this code after you write the mean function and all that, then the program will keep on printing the same message over and over again. And you cannot exit out of the program. If you are running a Linux system you can press control c, and the program will exit immediately. But here is an infinite loop; it executes an infinite number of times.

(Refer Slide Time: 02:01)



Break from loop!

- C allows one to explicitly break out of a loop by executing the statement break;
- When the break; statement is encountered, execution breaks out of the inner-most enclosing loop body (while, do-while, for or switch) and control goes to the statement after that loop body.
- Following program reads all numbers until a -1 is read and sums them (except the last -1).

```
int a, sum = 0;
while (1) {
        scanf("%d",&a);
        if (a == -1)  {
                break; }
        sum = sum +a;
}
printf("%d ", sum);
```

while (!(a == -1))

So, is there a statement which helps us to exit from a loop? Now, this is useful not just to handle infinite loops, even when you write normal loops it is important to have these constructs; they make your programming easier. So, c allows a programmer to explicitly break out of a loop using a particular statement known as break. When the break statement is encountered, the execution breaks out of the inner most loop. So, what is a loop? So far we have seen while loop, do-while loop, and for loop; later we will see a construct called switch.

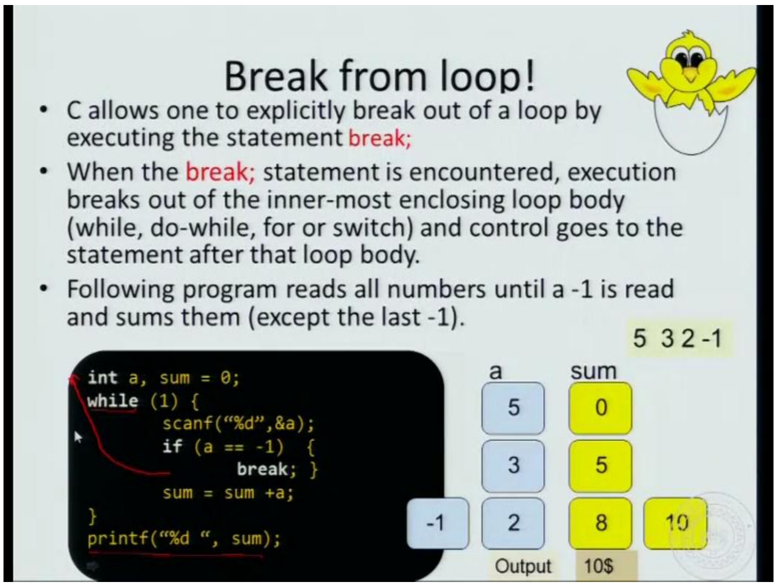
So, whatever is the inner most loop, notice that we have talked about double loops, we have talked about the while loop within a while loop, we have talked about a for loop within a for loop, whatever is the inner most for loop within which a particular break occurs, it will exit out of that.

So, let us write a very simple program which reads all numbers still minus 1 is seen and adds them up; minus 1 is excluded. So, you will, you can write a while loop; you have written this before where the while loops test condition was somewhat most sophisticated. Earlier we wrote something like while, if you recall, if you, we had written a loop saying, while not of a equal to minus 1. So, this was the earlier loop that we had written.

And in this case, let us write a similar program, but with a simpler test expression which is just while 1. So, you always enter the loop, no matter what number you read. So, initialize the variable sum to 0, declare the integer variable a, and then you enter the while loop because the test is true; you scan a number; and here is a use of the break statement.

If the scanned number is minus 1 you break out of the loop; if it is not minus 1, you go to the next statement which is sum equal to sum plus a. So, you add the number. Again you go back to the loop; the test condition is always true, so, you enter and read the next number. So, the net effect of the loop is that whenever you see a minus 1, it immediately exits out of a loop; otherwise, it adds that number to the loop.

(Refer Slide Time: 04:40).



Break from loop!

- C allows one to explicitly break out of a loop by executing the statement break;
- When the break; statement is encountered, execution breaks out of the inner-most enclosing loop body (while, do-while, for or switch) and control goes to the statement after that loop body.
- Following program reads all numbers until a -1 is read and sums them (except the last -1).

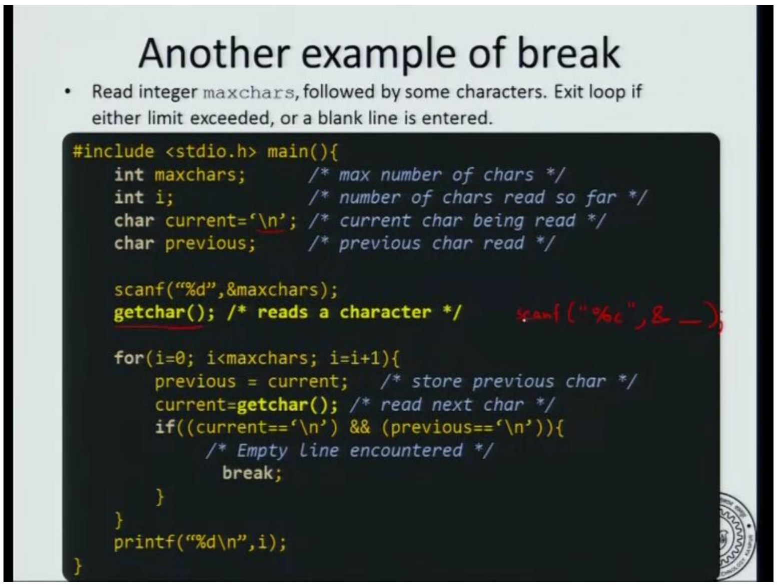
```
int a, sum = 0;
while (1) {
        scanf("%d",&a);
        if (a == -1)  {
                break; }
        sum = sum +a;
}
printf("%d ", sum);
```

So, let us look at using a sample input. Initially, a, is undefined; it is just declared. So, it has some garbage value. And sum is initialized to 0. Let us say that the input is 5, 3, 2, minus 1. While 1, so, 1 is true; therefore, you enter the while loop; you scan f, the first number. So, a, becomes 5; a, is not minus 1. Therefore, you go to sum equal to sum plus a. So, sum becomes 5.

And then you go back to the while loop; the test condition is still true, while 1. So, you read the next number 3; 3 is not minus 1. So, you add it to the sum; sum becomes 8. And you go back; the same thing occurs. So, you have the third number read which is 2; add it to the sum, and sum becomes 1. Then you read the next number; and now, a, is minus 1. So, what happens? The, if condition, the expression within the if statement is true; and you execute the statement inside the if condition, the statement is break.

So, recall that the rule of break says that exit out of the inner most loop. So, in particular, what is the inner most loop? You look, you starting from here, and imagine that you are going outwards towards the top of the program. The first loop that you will encounter on its way that is the loop that you will exit out of… In particular, break does not mean that exit out of the if condition; break means that exit out of the first loop that you see when you start from the statement and work outwards. So, that is this while loop. Break means you will exit out of that while loop and print this statement. So, you will print that the output is 10.

(Refer Slide Time: 07:02)



Another example of break

- Read integer `maxchars`, followed by some characters. Exit loop if either limit exceeded, or a blank line is entered.

```
#include <stdio.h> main(){
    int maxchars;       /* max number of chars */
    int i;              /* number of chars read so far */
    char current='\n'; /* current char being read */
    char previous;      /* previous char read */

    scanf("%d",&maxchars);
    getchar(); /* reads a character */        scanf("%c",& _);

    for(i=0; i<maxchars; i=i+1){
        previous = current;   /* store previous char */
        current=getchar(); /* read next char */
        if((current=='\n') && (previous=='\n')){
            /* Empty line encountered */
              break;
        }
    }
    printf("%d\n",i);
}
```

So, let us give in dealing with integers for a long time. Let us write a small program using characters. So, here is a problem, and let us say that we are writing a very simple editor. Now, the editor has the following property. There are a particular number of maximum characters that you can read; maybe it is 1000. So, you can typein a bunch of characters until one of the two conditions occur; either you enter a blank line by itself which is indicating that I am done entering the text or you enter more than the maximum number of characters available.

So, recall, there are 2 conditions for exiting out of our so called editor; you can type a lot of characters, if your limit was 1000 and you exit 1000 then you cannot typein any more characters, and you exit. Otherwise, if you are within 1000 characters but you entered a blank line that is indicating that you are done, you have nothing more to enter, then also you should exit. So, there are 2 conditions. Let us try to write this code.

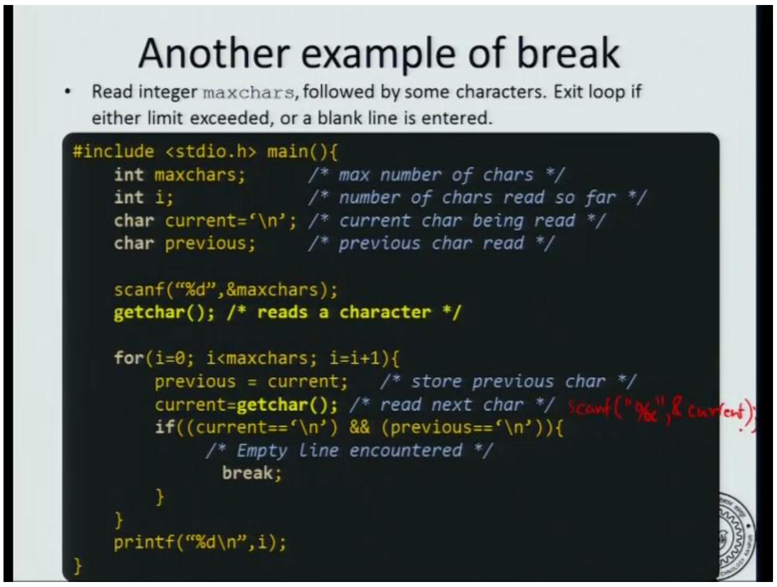
So, you have maximum characters. And let us say, I scan that. Then an, i, which counts how many characters I have read so far; so, i should initialize to 0. And then there is a current character, and then there is a previous character. So, I will initialize current to the new line character. Now, there is a particular reason for that which will become clear later. So, you should initialize current to a particular character.

And then what I do is, use the getchar function. So, getchar function reads a particular character from the input and stores it in some variable if you need to. Instead you can also say something like scan f percentage c, and some, into some variable. So, you can do either of these 2 things. And they are almost equal. So, you read one more character. Now, what should you do? You initialize by starting from 0. So, you have read no characters until now. And until you read maximum member of characters, so you execute this loop.

Remember that I said that for loop is good when you know the number of iterations in advance. So, we know that atmost we will execute maximum number of character times because that is the maximum number of characters we are allowed to field. So, for loop is slightly better than a while loop. You can also do it using a while loop if you want. So, you say for i equal to 0, i less than maximum characters, i equal to i plus 1. Now, we will do this programming style that we should be familiar with right now. So, previous becomes current and current becomes the next character; so, previous equal to current.

So, this will store the current character into the variable previous. Then you read the next character using getchar.

(Refer Slide Time: 10:52).



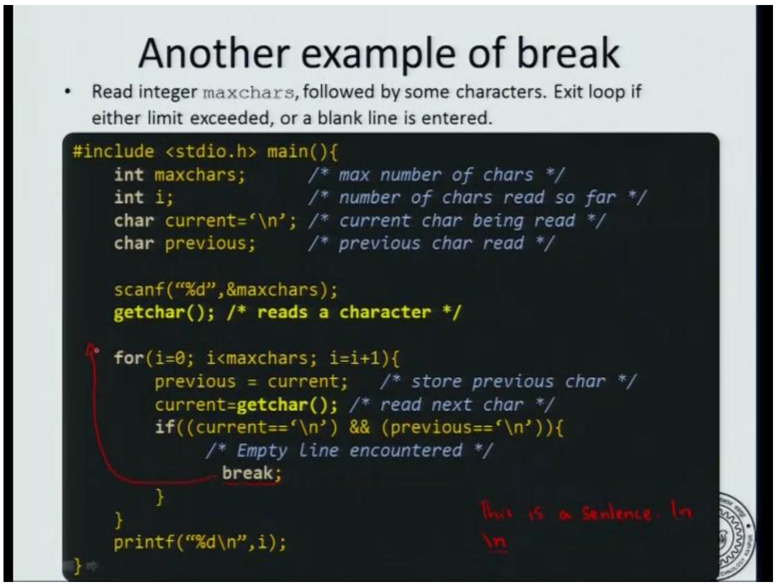
Another example of break

- Read integer maxchars, followed by some characters. Exit loop if either limit exceeded, or a blank line is entered.

```
#include <stdio.h> main(){
    int maxchars;       /* max number of chars */
    int i;              /* number of chars read so far */
    char current='\n'; /* current char being read */
    char previous;      /* previous char read */

    scanf("%d",&maxchars);
    getchar(); /* reads a character */

    for(i=0; i<maxchars; i=i+1){
        previous = current;    /* store previous char */
        current=getchar(); /* read next char */  scanf("%c",&current);
        if((current=='\n') && (previous=='\n')){
             /* Empty line encountered */
               break;
        }
    }
    printf("%d\n",i);
}
```

And as I said before, you can also write equivalently scan f percentage c and current. So, both these are almost equivalent that is a slight difference, but we will it is not important as of now.

(Refer Slide Time: 11:27)



Another example of break

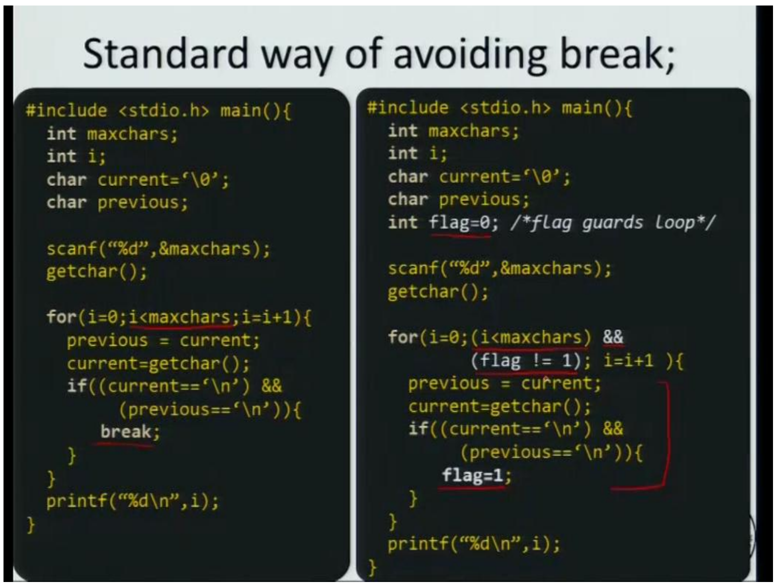
- Read integer maxchars, followed by some characters. Exit loop if either limit exceeded, or a blank line is entered.

```
#include <stdio.h> main(){
    int maxchars;       /* max number of chars */
    int i;              /* number of chars read so far */
    char current='\n'; /* current char being read */
    char previous;      /* previous char read */

    scanf("%d",&maxchars);
    getchar(); /* reads a character */

    for(i=0; i<maxchars; i=i+1){
        previous = current;    /* store previous char */
        current=getchar(); /* read next char */
        if((current=='\n') && (previous=='\n')){
             /* Empty line encountered */
               break;
        }
    }                                          This is a sentence. \n
    printf("%d\n",i);                          \n
}
```

Now, if current is new line and the previous was new line, so, when will that happen? Suppose I write this is a sentence, I will explicitly represent the new line. So, when I press enter I will have a new line character here. And when will a blank line occur? When the next character is also new line. So, by a blank line what I mean is that the current sentence is over, so I press a new line; and the next character on the next line is also a new line; that is what is actually meant by a blank line.

So, when that happens then we know that an empty line has been encountered; and here is the important thing break because one of the conditions to exit out of that loop was that either at maximum number of characters is encountered or a blank line is encountered. So, you may not have encountered maximum number of characters, but you have encountered a blank line. So, you should exit out of the proof, exit out of the loop. Again the rule is that break out of the inner most for loop, inner most loop, which in this case is just for loop. So, you get out of that loop and printf a new line.

(Refer Slide Time: 12:44)



Standard way of avoiding break;

```
#include <stdio.h> main(){
  int maxchars;
  int i;
  char current='\0';
  char previous;

  scanf("%d",&maxchars);
  getchar();

  for(i=0;i<maxchars;i=i+1){
    previous = current;
    current=getchar();
    if((current=='\n') &&
        (previous=='\n')){
      break;
    }
  }
  printf("%d\n",i);
}
```

```
#include <stdio.h> main(){
  int maxchars;
  int i;
  char current='\0';
  char previous;
  int flag=0; /*flag guards loop*/

  scanf("%d",&maxchars);
  getchar();

  for(i=0;(i<maxchars) &&
        (flag != 1); i=i+1 ){
    previous = current;
    current=getchar();
    if((current=='\n') &&
        (previous=='\n')){
      flag=1;
    }
  }
  printf("%d\n",i);
}
```

Now, as with many constructs in c, you can avoid break all together. You can write code if you have used break, you can right equivalent logic without using break. So, here is a standard way to do it. So, here is the code that we just delt with. It had 2 exit conditions - one is that the number of characters that you read is greater than the maximum allowed; another exit condition was that you had entered a blank line. So, here we used the break statement.

And now I want to write an equivalent loop without using the break statement. And here is a very standard programmatic style. These are known as flags. So, flag is just a variable which indicates that a particular condition has occurred. Initialize flag to just 0.

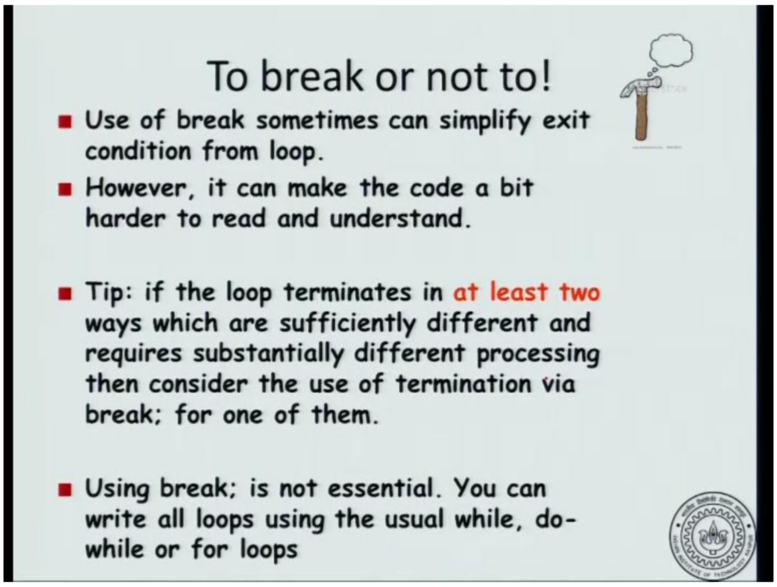
In our code, what flag is supposed to do is that it will indicate whether a blank line has occurred or not. So, let us first look at the body of the loop; without looking at the loop head first. Let us just look at the body of the loop. So, it is similar to what went before. Instead of the break statement, what I will do is, if I realize that an empty line has happened then I will set flat to 1; notice that flag was initially 0.

So, flag equal to 1 will indicate that an empty line has been seen. Now, I will modify the loop as follows. Remember that the test condition here is just that maximum number of characters has occurred. Instead, I will check for 2 conditions in the for loop. I will check that maximum number of characters have not occurred, and I will also check that flag is not 1 because flag is 1 means that a new line, a blank line has been encountered. So, I will check for both these conditions in the for loop itself.

If either of them is true that is; sorry, if either of them is false that is if i is greater than or equal to maximum characters, or flag equal to 1, then the test condition will become false and you will exit out of the loop. So, here is a standard way to avoid a break. And notice that this condition is negated in the for loop because the condition in the for loop is the condition for entering the loop. So, to exit out of the loop you need flag equal to 1.

So, in summary, what I want to say is that if you want to write a code using break, you can also write it without using break. One of the standard way to do it is by using a flag variable for whatever condition that we want to check. You can pick either of this style whichever suits you more.

(Refer Slide Time: 16:35)



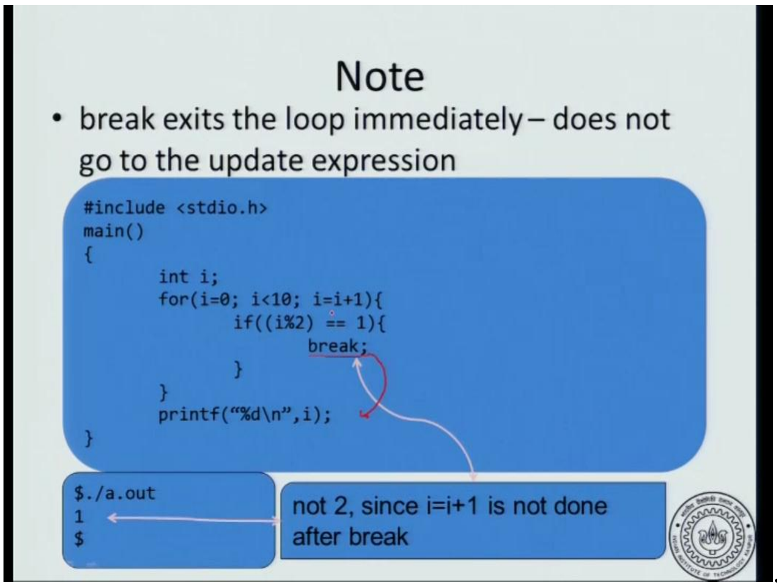
To break or not to!

- Use of break sometimes can simplify exit condition from loop.
- However, it can make the code a bit harder to read and understand.
- Tip: if the loop terminates in at least two ways which are sufficiently different and requires substantially different processing then consider the use of termination via break; for one of them.
- Using break; is not essential. You can write all loops using the usual while, do-while or for loops

So, how do we decide whether to use the break statement or not? Sometimes the use of the break statement can simplify the exit condition. And on the other hand, it could also make the code a bit harder to read. What do I mean by harder to read? When I see the for loop in the code on the right hand side, it is clear that there are 2 ways to exit out of the for loop - one is i greater than or equal to maximum characters, the other is flag equal to 1. Just by looking at the for loop, I can say that, ok, here are the 2 conditions for which the loop will terminate - i greater than are equal to max chars, or flag equal to 1.

On the other hand, if you look at this left hand side code, I actually have to look at the body of the code to realize what are the ways of exiting out of loop. So, you have to understand the body of the loop in order to see what are the conditions for the loop to exit. It is not just i greater than or equal to max chars. So, in that sense, the code with break is harder to understand than the code without break. It still recommended to use break when you have 2 or more exit conditions out of a for loop. So, typically programmers do use break and it is just a matter of style whether you we will use break or not; I myself prefer using a break.

(Refer Slide Time: 18:07)



Note

- break exits the loop immediately – does not go to the update expression

```
#include <stdio.h>
main()
{
        int i;
        for(i=0; i<10; i=i+1){
                if((i%2) == 1){
                        break;
                }
        }
        printf("%d\n",i);
}
```

```
$./a.out
1
$
```

not 2, since i=i+1 is not done after break

One final thing about the break statement; when you use break statement initially, it is important to notice that break causes an exit immediately out of the loop. So, remember when you have a for loop, the normal execution order is you initialize, then you test. So, this is step 1, this is step 2, then you execute the body of the loop that step 3, and then you update this is step 4, and then go back to the test condition. So, this is the normal execution order of the loop.

When you encounter a break, you exit immediately out of the loop. In particular, when you break you do not go back to the update statement. So, let us examine what this code will do? You have, i equal to 0, i less than 10 increment i. So, you start with i equal to 0; i modulo 2 will be 0 modulo 2 which is 0. So, it will say, ok fine, you need not get into the if condition.

Then i equal to i plus 1; so, i equal to 1; 1 is less than 10; you enter the for loop; 1 modulo 2 is 1; so, you will break. When you break you immediately get out of a loop. So, when you print this then i will be 1. So, in particular, i is not 2, which is what will happen if you go back and update i equal to i plus 1, before exiting out of the loop. So, the important thing to notice is that it is not 2, since i equal to i plus 1 is not done when you break. When you break you get out the loop immediately without doing the update state.