

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

Lecture – 24
Example of Loops (Contd.)

So in the last lecture we have seen how we can write a program to find out whether it is prime or not.

(Refer Slide Time: 00:19)

Example 1: Test if a number is prime or not

```
#include <stdio.h>
main()
{
    int n, i=2;
    scanf ("%d", &n);
    while (i < n) {
        if (n % i == 0) {
            printf ("%d is not a prime \n", n);
            exit;
        }
        i++;
    }
    printf ("%d is a prime \n", n);
}
```

n
13
 $n-2$
 \sqrt{n}

i
2
3
4
5
6
7
8
9
10
11
12

11 times

Now, this program that you see that we have we have discussed in the last class, we will certainly give us correct result. Now our job is not only to write a correct program it is also very stimulating intellectually to think of how we can make it a more efficient, program often the program can be correct, but it is not very efficient. What do I mean by this efficiency? Say if we look at this program and again I take the earlier example that suppose I have my n is 13.

Now, and I start with the value of i to be 2 and n is 13 and since it is prime I am checking this condition time and again and since this is third I mean prime number how many times it will never come here, it will never succeed. So, I will have to carry on this loop and ultimately when I exhaust all the possibilities then only I will come to this point.

So, how many times do I have to do this let us see, I am starting with 2 checking ones then start incrementing and making it 3, then 4 then testing with 5, then testing with 6, testing with 7, testing with 8, testing with 9 10 11 12. So, how many times did I check it 1 2 3 4 5 6 7 8 9 10 11 times, I had checked it with 11 times

Now, do I really need to check it with 11 times? If you think a little bit if this number is n then, I did not check it and I am starting with 2. So, n minus 2 times I am checking right do I really need to check n minus 2 times. If I do not find it divisible by squared within square root of n times. So, square root of 13 as an integer what would it be? 13 would be 3 point something right. So, 4 times within 4 if it is not divisible, it will not be divisible later by the future numbers also right.

So, if I just apply this knowledge that if it is not divisible within square root of n , then it will not be divisible later I can make this program much more efficient how let us see, let us look at the and a more efficient version of the program.

(Refer Slide Time: 03:48)

```
#include <stdio.h>
main()
{
  int n, i=3;
  scanf ("%d", &n);
  while (i < sqrt(n)) {
    if (n % i == 0) {
      printf ("%d is not a prime \n", n);
      exit;
    }
    i = i + 2;
  }
  printf ("%d is a prime \n", n);
}
```

So, here you see I am starting with 3 and I am reading the value of n fine. So, far no problem I am starting with n .

Now, here I am checking not up to n I am checking I mean in the earlier case what we did is sorry earlier case what we did is we tested with up to n , but here what we are doing is I am testing with up to square root of n and if n is divisible by i ; that means, say 13, I

first try with 3 it is not divisible is try with 4 and. So, square i is for that is less than the greater than the square root of n right. So, with that I will come out and every time what I am doing is I am implementing i by 2. So, I am testing with 3, I am testing with 5, I am testing with 7 like that I am going on. If I apply this then I can always make this program more efficient clear.

(Refer Slide Time: 05:48)

The slide is titled "More efficient??" and contains the following C code snippet:

```
#include <stdio.h>
main()
{
    int n, i=3;
    scanf ("%d", &n);
    while (i < sqrt(n)) {
        if (n % i == 0) {
            printf ("%d is not a prime \n", n);
            exit;
        }
        i = i + 2;
    }
    printf ("%d is a prime \n", n);
}
```

Handwritten annotations on the slide include:

- A blue checkmark next to `i=3;`
- A blue checkmark next to `printf ("%d is not a prime \n", n);`
- A blue checkmark next to `i = i + 2;`
- A diagram on the right side showing the variable `i` with values 3 and 5. A blue arrow points from the value 3 to the number 21, and another blue arrow points from the value 5 to the number 23.

So, you see by the. So, let us take another example here say I am trying to find out whether 21 is a prime number or not. So, I start with i to be 3. So, 3 is of course, 21 will not work sorry let me change it 21 is not a prime. So, first time in first short it will give me that it is not the prime fine, but if instead I had a 23 then what will happen? I start with 3, it does not divide then I come and make it 5. If something is not divisible by 3 it will not be divisible by I mean next square root of that I am just going by 1 more I come to 5 it is not divisible by that, what is the square root of 23? It should be 4 point something. So, again quickly I am coming to the point where I can see that it is not divisible. So, by doing this using this small technique, I can make it a little faster the number of checks will be much more reduced.

So, now, let us go to another example.

(Refer Slide Time: 07:04)

Example 2: Find the sum of digits of a number

```
#include <stdio.h>
main()
{
    int n, sum=0;
    scanf ("%d", &n);
    while (n != 0) {
        sum = sum + (n % 10);
        n = n / 10;
    }
    printf ("The sum of digits of the number is %d \n", sum);
}
```

Handwritten calculations for 123 and 243:

$$\begin{array}{r} 123 \\ +2 \\ +3 \\ \hline 6 \end{array}$$
$$\begin{array}{r} 243 \\ +4 \\ +3 \\ \hline 9 \end{array}$$

This is again finding the sum of and sum of the digits of a number. Now this 1 we did not do sum of the digits of a number what is meant by this? Suppose I have got a number 123, sum of the digits of a number means 1 plus 2 plus 3 6. If my number is 243 then the sum of the digits will be 2 plus 4 plus 3 6 and 3, 9 this is what I want to find out.

So, how do you go about doing that, how do I find out the digits? Say 1 2 3 from there how do I extract out 1 2 and 3.

(Refer Slide Time: 08:05)

Example 2: Find the sum of digits of a number

```
#include <stdio.h>
main()
{
    int n, sum=0;
    scanf ("%d", &n);
    while (n != 0) {
        sum = sum + (n % 10);
        n = n / 10;
    }
    printf ("The sum of digits of the number is %d \n", sum);
}
```

Handwritten calculations showing the extraction of digits from 123:

$$\begin{array}{r} 10 \overline{) 123} \quad (12 \\ \underline{120} \\ 3 \end{array}$$

3 → 3

$$\begin{array}{r} 10 \overline{) 12} \quad (1 \\ \underline{10} \\ 2 \end{array}$$

2 → 2

$$\begin{array}{r} 10 \overline{) 1} \quad (0 \\ \underline{0} \\ 1 \end{array}$$

1 → 1

Sum: 3 + 2 + 1 = 6

Just think over if I have got 123 and I divide it by 10, I will get a quotient 12 and a remainder. So, the remainder is 1 digit, now I take the quotient divided by 10, I will get a remainder sorry a quotient and the remainder, the remainder is another digit and then I go on till it is the number is not equal to 0. So, next I take one and divide it by 10, the quotient is 0 and the remainder is 1.

So, you see I have got all the digits from the right hand side one after another right. So, I can add them that is my basic way. So, let us keep it on the side and see what we have done in this program. I have got a number n, now this number that has been given initially 123, I do not need to preserve that number. So, I can play with that and I have to make a sum of this right 1 2 and 3. So, I keep a sum to be initially 0. I keep a variable sum which is initialized to 0 as we saw in the earlier examples and sum is also an integer then I read the number that is say 123 has come.

Now, while the number n that is 123, 123 is not zero then I take add n modulus 10 modulus 10 will give me what? The remainder 3 and add it to the sum. So, my sum now becomes 3, and then I want to find out this thing also. I make my n to be n divided by 10 now this operation gives you the quotient right this operation gives you the remainder.

So, I now I get n by 10 means now 12, and I go back to this loop I check that 12 is not zero. So, I will again do that, sum will be sum what was that 3 plus n modulus 10. So, I again divided by 10 and take the remainder and that remainder is added sum plus this. So, 3 plus 2 will be 5 and then I divide n by 10. So, now, I find out 1 still I go back 1 is not I go back actually here, 1 is not zero sure. So, I again divide it by I take the remainder divided by 10 and take the remainder. So, the remainder is 1, I added with the sum. So, it will be 6 and then I divide I find the quotient and that is 0, I go back again here and find that n is not equal to 0 condition is not true. So, I come out of the loop and I apply this printf statement what is the printf statement doing? The sum of digits of the given number I cannot say given number n because n has already changed I will come to that in a moment.

Now, it is sum. So, I have got the sum. So, this is another example of while loop now my question is if you have understood it, I want the output to come as printf; the sum of digits of the number 123 is 6 how can I do what modification should I do in this program.

(Refer Slide Time: 13:02)

Example 2: Find the sum of digits of a number

The sum of digits of 123 is 6

```
#include <stdio.h>
main()
{
    int n, sum=0;
    scanf ("%d", &n);
    while (n != 0) {
        sum = sum + (n % 10);
        n = n / 10;
    }
    printf ("The sum of digits of the number is %d\n", sum);
}
```

123 → 12
↓
1
↓
0

num = n
%d num

What I want to be printed is the sum of digits of 123 is 6 what should I modify? Of course, I should modify this statement, this statement should be the sum of the digits of the number percentage d is percentage d, and here before sum what should I do? My n that was provided by the user has already been destroyed in this loop.

So, what because every time I am dividing it by 10 and making a new number, 123 is becoming 12 and then it is becoming 1 then is becoming 0. So, I have to save it somewhere this 123. So, what I can do here after this reading this, I can save it in another variable num has n and here I will put num comma now because I am destroying n here, but I am not touching num. So, num will remain intact only another thing that I have to do I have to add this num here I must declare it num as an integer otherwise it will give you syntax error all right.


So, this is another nice example of while using which we can find the number of digits the sum of the number of digits of a number.

(Refer Slide Time: 15:04)

Example 3: Decimal to binary conversion

```
#include <stdio.h>
main()
{
    int dec;
    scanf ("%d", &dec);
    do
    {
        printf ("%2d", (dec % 2));
        dec = dec / 2;
    } while (dec != 0);
    printf ("\n");
}
```

4	100
5	101
1	001
2	010



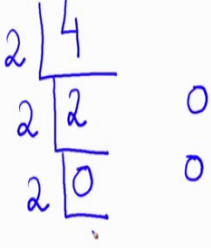
With that we move to another example; decimal to binary conversion some of you may know it and some of you may not be very conversant with that. We know that we have got the decimal binary number system where the base is 2. So, everything is expressed using zeros and one.

So, sorry if I have a binary digit sorry decimal digit 4, it is binary equivalent is 1 0 0, if we have 5 that is 1 0 1, if it is 1 then it is 0 0 1 or just 1, if it is 2 it is 0 1 0 like that the question is that given our decimal digit how can I convert it to binary what is the algorithm? The algorithm is something like this say I take 4 and divide it by 2.

(Refer Slide Time: 16:23)

Example 3: Decimal to binary conversion

```
#include <stdio.h>
main()
{
    int dec;
    scanf ("%d", &dec);
    do
    {
        printf ("%2d", (dec % 2));
        dec = dec / 2;
    } while (dec != 0);
    printf ("\n");
}
```



So, the remainder is 2 a sorry the quotient is 2 and the remainder is 0 right and then I divide it by 2 again, the quotient is 0 and this is 0 now when I divide it by 2 further what will happen [FL] [FL] [FL][FL]

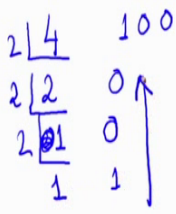
[FL]

[FL]

(Refer Slide Time: 17:50)

Example 3: Decimal to binary conversion

```
#include <stdio.h>
main()
{
    int dec;
    scanf ("%d", &dec);
    do
    {
        printf ("%2d", (dec % 2));
        dec = dec / 2;
    } while (dec != 0);
    printf ("\n");
}
```



[FL] [FL] yeah [FL].

(Refer Slide Time: 18:36)

Example 3: Decimal to binary conversion

```
#include <stdio.h>
main()
{
    int dec;
    scanf ("%d", &dec);
    do
    {
        printf ("%2d", (dec % 2));
        dec = dec / 2;
    } while (dec != 0);
    printf ("\n");
}
```

100

Let us take the number 4 and I want to find out what is the binary equivalent of this, the algorithm goes like this I first divide it by 2, because 2 is the base of any binary system I divide it by 2. So, I get the quotient as 2 and remainder as 0.

Next I divide it again by 2. So, my quotient is 1 and remainder is 0. Next I divide it again by 2 my quotient is 0 and remainder is one. So, I go on dividing till I get a quotient to be 0 and I have remembered all the remainders that I got now I can get the binary if I did it in this direction. So, it is 1 0 0 all right.

(Refer Slide Time: 19:53)

Example 3: Decimal to binary conversion

```
#include <stdio.h>
main()
{
    int dec;
    scanf ("%d", &dec);
    do
    {
        printf ("%2d", (dec % 2));
        dec = dec / 2;
    } while (dec != 0);
    printf ("\n");
}
```

2^3	2^2	2^1	2^0
8	4	2	1
1	1	1	1



Let us take another example suppose I want to have the number 15, now what is the binary of 15 to understand that you let us look at the weights of the different positions in binary, this one is with 2 to the power 0, 2 to the power 1, 2 to the power 2, 2 to the power 3; that means, this is 8 4 2 1 now 15 if I have to have I must have a 1 here and 7 more. So, 1 here 8 and 4 12 and 1 here 12, 13, 14 and 15 all this should be 1.

So, can we find out find this out let us see 15 I divide by 2 my quotient is 7 and remainder is 1. I again divide it by 2 my quotient is 3 and remainder is 1. I divide it by 2 again my quotient is 1, remainder is 1, I divide it by 2 again my quotient is 0 and the remainder is 1, I did it in this direction and we get 1 1 1 1 all right.

(Refer Slide Time: 21:22)

Example 3: Decimal to binary conversion

```

#include <stdio.h>
main()
{
  int dec;
  scanf ("%d", &dec);
  do
  {
    printf ("%d", (dec % 2));
    dec = dec / 2;
  } while (dec != 0);
  printf ("\n");
}

```

Handwritten diagram showing the conversion of 23 to binary:

23	
11	1
5	1
2	1
1	0
0	1

Weights: 2^3 16, 2^2 4, 2^1 2, 2^0 1

Binary result: 1 0 1 1 1

while ()

Let us take another example again the weights of the system are 1 2 4 8. 2 to the power 0 2 to the power 1, 2 to the power 3, 2 to the power sorry 2 to the power 2, 2 to the power 3 now let us take a number and next one will be 16 is 2 to the power 4.

Now, let us take a number 23, what will be the binary representation of that let us see here 16 will be there of course, I have greater than 16. So, 23 16 and 6 and 7 more. So, this should be 0 I cannot put a 1 here, because there to be 16 and 8 will be 24. So, 1 here 20 this this. So, my binary is 1 0 1 1 1 let us try in our algorithm divide it by 2, my quotient is 11, I have got a remainder 1 I divide it by 2 my quotient is 5 remainder is 1, I divide it by 2 quotient is 2 remainder is 1 divided by 2 is 1 and 0 and divided by 2 is 0 and 1. So, I reach a 0 I stop and this is the pattern 1 0 1 1 1, 1 0 1 1 1 all right.

Now, let us see how this algorithm that we are talking of can be encoded in C program. You see here I have declared 1 variable as dec; that means, the decimal number that I will be reading. So, I am reading that number here. Now in the earlier example we did it while do now here we are doing do while we are doing something and then I am putting the while condition later. Do what am I doing printf this means percentage 2 d means I am printing in binary, I mean this I am taking this dec and I mean 1 beta I am printing I am defining out the remainder I find out the remainder and then divide it.

And I go on doing this I divide it and I go on doing this while this is the number the remainder is not equal to 0. I am dividing it by 2 and finding out the new number, that this is this point what am I doing? I am finding out the quotient, here I am finding out the quotient and I am based on the quotient I am going on till I get this 0. Do not bother about this I can simply take the this thing, first this thing and I go on printing it.

So, here what I am printing I am printing in this in this format I am printing in this format let us forget about this for the time being all right let us put percent d then I am dividing it like this way all right. I divide by 2 continually and I am printing in this way 1 then I change it to 11 and then divide it then dec becomes 5, I print it and go on doing it. So, at least once I am dividing till it is 0. This is the algorithm for decimal to binary conversion you can also try to write it not with do while, but just by while you can I leave it as an assignment for you to write it using while some condition and not do while not do while.

(Refer Slide Time: 25:57)

Example 4: Compute GCD of two numbers

```
#include <stdio.h>
main()
{
    int A, B, temp;
    scanf ("%d %d", &A, &B);
    if (A > B) { temp = A; A = B; B = temp; }
    while ((B % A) != 0) {
        temp = B % A;
        B = A;
        A = temp;
    }
    printf ("The GCD is %d", A);
}
```

*B = 45
A = 12*

12) 45 (3
 36

 9) 12 (1
 9

 3) 9 (3
 9

 0

Initial: A=12, B=45
Iteration 1: temp=9, B=12, A=9
Iteration 2: temp=3, B=9, A=3
B % A = 0 → GCD is 3.

So, next let us move to another example that is also very interesting and known to us that is finding the greatest common divisor of 2 numbers what is the algorithm? The algorithm is shown here, suppose I have got a number 45 and 12, I am to find out the greatest common divisor or the hcf of these 2 numbers. What I do is I divide the bigger number by the smaller number and get a remainder. You see remainders are becoming smaller. So, important they come to the remainder

And then if the remainder is not zero, I divide now the smaller number by this remainder and then I get again a remainder which is non-zero, then I take the divisor as the current divisor and divide it by this remainder until we get 0 that is what we used to do in school right. So, ultimately when I get 0 whatever is the divisor currently that is our hcf or gcb. Here you see how the code is read you just look at the code a little bit and try to understand it.

A and B are 2 numbers, now I do not know as to which one is smaller and which one is bigger. Here in our example we can see one is 12, one is 45, but one is 12 one is 45, but I am going to try this here the computer does not know which one is 12 which one is 45 and I am taking another variable temp I am reading the numbers A and B. Now if A is greater than B; that means, I am taking the greater number and moving that into temp out of this 12 and 45 which one is the greater number 45.

So, temp becomes 45 and b becomes a B is the smaller number. So, 12 becomes c and b becomes temp, now you see I have written these 3 in 1 shot. So, let us see here what is happening after this first temp is 45, A is 12 and B is 45 now what I am doing here I am keeping this temp what I am doing here I am dividing B by A. So, 45 is being divided by A 12 and I am checking the remainder the remainder is not zero you can see that. Since it is not zero I am taking temp to be the modulus of 12 sorry b is what? B was 45 right and A was 12 by this point. So, then and temp was 45 I try this it is not zero.

So, then I take this remainder 9 and again make that to be b. So, now, B becomes 9 and A becomes the temp which was this number and I go on dividing this and I go on carrying out this loop or till this becomes 0. As soon as this becomes 0 then I have got my GCD stored in A, because every time I am taking the A in A I am storing the temp and what is temp? Temp is after I divide I get the remainder and that I am storing in temp and that temp is coming in A all right it is being stored in a and that is also copied in B, but in the next loop that is being divided by. So, here you are having 3 9 and here 3. So, look at this iteration 1, here looking at this picture first temp will become 9, B is 12 and A is 9.

Next time temp is becoming 3, B is 9 this 9 and A is becoming 10 and whenever I find that B divided by A this part is 0 remainder is 0 then that is my GCD A is my GCD, I get the 3 there is another very interesting example of computing using this while loop and repeatedly I am doing this till I find the remainder to be 0. So, there are many such nice examples and we will give you some more during for assignments, which we will have to solve and get more you will get more confidence about it. So, next we look at we will come to some of the pitfalls of loop, and then move to something more useful something very useful that is the concept of arrays ok.

Thank you.