CS1100 – Introduction to Programming

Trimester 3, April – June 2021

Instructor:

Shweta Agrawal (shweta.a@cse.iitm.ac.in)

Lecture 16

More examples of loops

We will study more examples of loops, especially nested loops.

- Printing patterns

More examples of loops

We will study more examples of loops, especially nested loops.

- Printing patterns
- Printing first k primes

More examples of loops

We will study more examples of loops, especially nested loops.

- Printing patterns
- Printing first k primes
- Finding prime factorization

More examples of loops

We will study more examples of loops, especially nested loops.

- Printing patterns
- Printing first k primes
- Finding prime factorization
- Printing staircase of numbers

More examples of loops

We will study more examples of loops, especially nested loops.

- Printing patterns
- Printing first k primes
- Finding prime factorization
- Printing staircase of numbers
- Computing positive square root of an integer, approximately

Printing patterns

```
********
********
********
********
```

```
#include<stdio.h>
main() {
    for (int i=1; i<=4; i++) {
        for (int j=1; j <=8; j++) {
            printf("*");
        }
        printf("\n");
    }
}
```

Printing patterns

```
**
****
******
********
```

```
#include<stdio.h>
main() {
    int k = 2;
    for (int i=1; i<=4; i++) {
        for (int j=1; j <=k; j++) {
            printf("*");
        }
        printf("\n");
        k = k+2;
    }
}
```

Printing first k primes

```
int n = 2;
while (count <= 10) {

    // decide if n is prime
    // if n is prime, increment counter, print n
    // irrespective if that increment n
}
```

Printing first k primes

```
int n = 2;
while (count <= 10) {
    // decide if n is prime
    int i = 2; int flag = 0;
    while (i < n) {
        if (n % i == 0) {
           flag = 1; break;
        }
        i = i+1;
     }
     // if n is prime, increment counter, print n
     if (0 == flag)  {
        printf("The %d prime is %d\n", count, n);
        count++;
     }
     // irrespective if that increment n
     n++;
}
```

Printing first k primes

```
int count = 1;  int n = 2;
while (count <= 10) {
    int i = 2; int flag = 0;
    while (i < n) {
        if (n % i == 0) {
           flag = 1; break;
        }
        i = i+1;
     }
     if (0 == flag)  {
        printf("The %d prime is %d\n", count, n);
        count++;
     }

     n++;
}
```

A note on design : Finding prime factors and their powers

Given n, test if it is prime. If not prime, print its prime factors with corresponding powers.

Idea

- Assume n is not prime.
- for i = 2 to n-1
  - detect if i is prime.
  - if i is prime, find largest power of i which divides n.
  - print i and the corresponding power.

A note on design : Finding prime factors and their powers

Given n, test if it is prime. If not prime, print its prime factors with corresponding powers.

Idea2

- Assume n is not prime.
- for i = 2 to n
  - ~~detect if i is prime.~~
  - ~~if i is prime,~~ find largest power of i which divides n.
  - print i and the corresponding power.
  - modify n.

Idea2 is simpler (to code). Needs thinking before coding.
Spend at least 5 minutes thinking on how to code.

Finding prime factorization

```
int n; scanf("%d", &n);
for (int i=2; i<= n; i++) {
    int count = 0;
    while (n % i == 0)    {
       count++; n=n/i;
    }
    if (count > 0 ) {
       printf("%d %d\n", i, count);
    }
}
```

Printing Staircase of Numbers

```
1
22
333
4444
```

- Accept input `n` ≥ 1 from user.
- Print a staircase containing `n` rows.
- Row 1 has a single 1, row 2 has two 2's and so on.
- Row n has n times the number n.

Use the do while construct

Printing Staircase of Numbers

```
1
22
333
4444
```

```
#include<stdio.h>
main() {
    int x;
    scanf("%d", &x);
    int i=1;
    do {
        int j = 1;
        do {
            printf("%d", i);
            j++;
        } while (j<=i);
        printf("\n");
        i++;
    } while (i<=x);
}
```

Compute positive square-root of an integer – approximately

For example $\sqrt{2}$ $\sqrt{102}$ $\sqrt{555}$ . . .

We have the time tested `sqrt` function – use that!

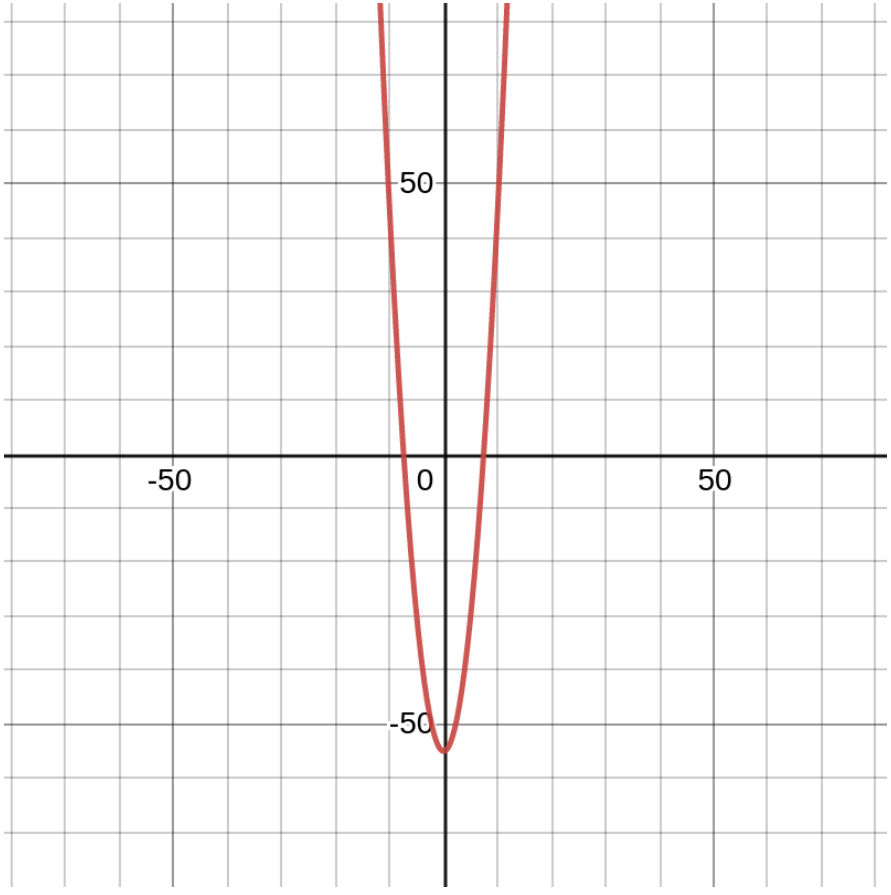
but how is that implemented?

We will study a simple and effective method – bisection method

Computing positive square root of a positive integer

Lets compute $\sqrt{55}$.

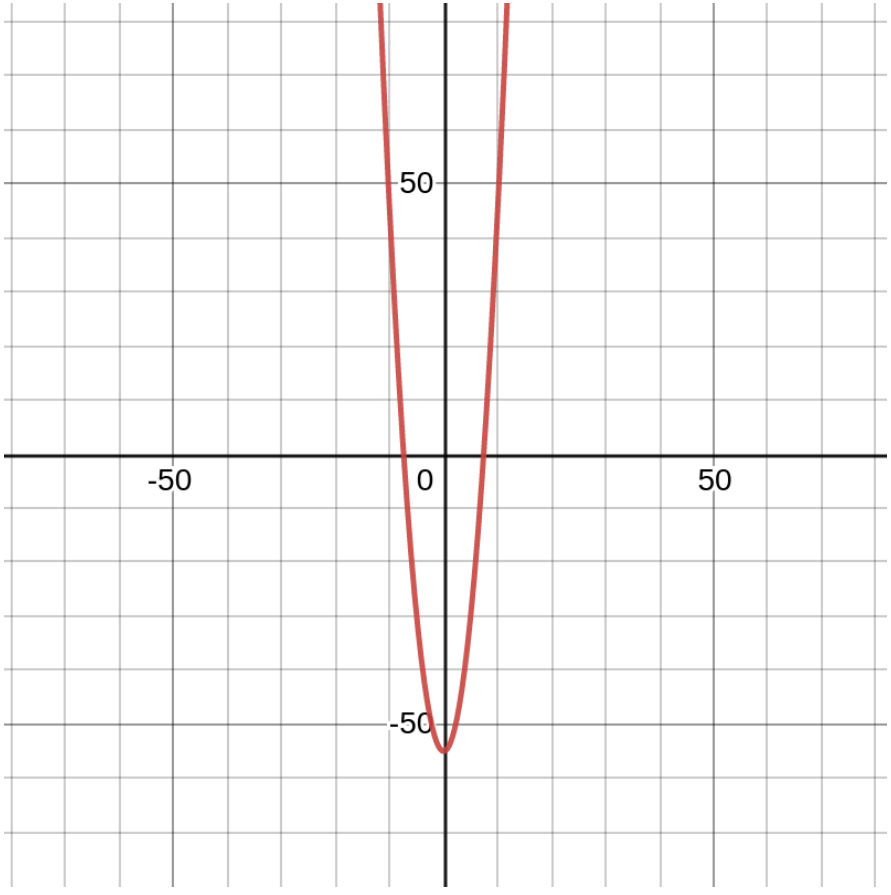
In fact we are interested in the value at which the function $f(x) = x^2 - 55$ evaluates to zero!



Computing positive square root of a positive integer

Lets compute $\sqrt{55}$.

In fact we are interested in the value at which the function $f(x) = x^2 - 55$ evaluates to zero!

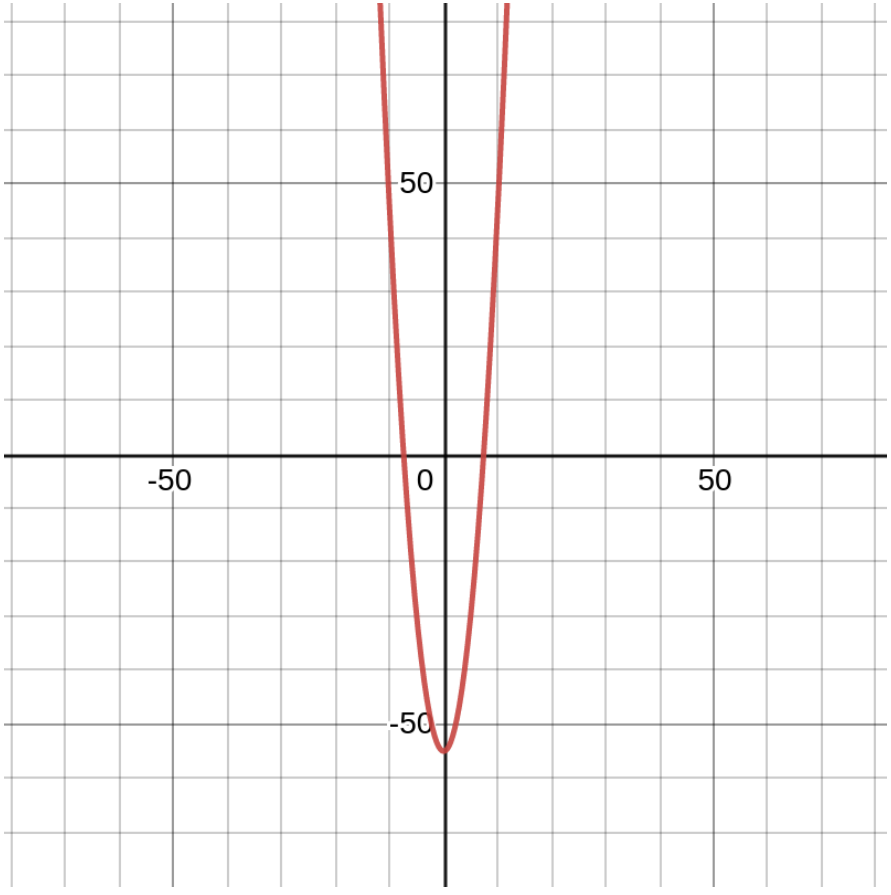


- Start with some initial guess : say 1. The value of $f(1)$ is -ve!
- Pick another guess where value is positive say 25.
- The function must be zero in between these two values.
- Keep refining your guess!

Computing positive square root of a positive integer

Lets compute $\sqrt{55}$.

In fact we are interested in the value at which the function $f(x) = x^2 - 55$ evaluates to zero!



- Two initial values xL $= 1$, xR $= 25$.
- How do we pick the refined guess?
  Take mid-point
- We now have 3 values xL, xR, xM.
- Which are useful?
  The two closest ones with opposite sign for $f(x)$.

Computing positive square root of a positive integer

- Two initial values such that $f(xL)$ is negative and $f(xR)$ is positive.
- Take mid-point $xM = \frac{xL+xR}{2}$.
- Pick two of xL, xR, xM which are closest and have opposite sign for $f(x)$.
- How long? Till the two estimates are close enough!

Computing positive square root of a positive integer

```
#include<stdio.h>
main() {
     double xL = 1; double xR = 25;
     double xM, epsilon;
     epsilon = 0.0001;

     while (xR - xL >= epsilon) {
          xM = (xL + xR) / 2;
          if ((xM * xM - 55) > 0) {
             xR = xM;
          } else {
             xL = xM;
          }
     }
     printf ("sqrt of 55 is %.4f\n", xL);
}
```