Introduction to Programming

Week – *2*, Lecture – *4*

Introduction to C – Part 2

SAURABH SRIVASTAVA

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

IIT KANPUR

The problems with our solver !!

First, we could only solve one specific equation – $x^2 + 3x - 15 = 0$

- … because we “fixed” the values of `a`, `b` and `c`

The problems with our solver !!

First, we could only solve one specific equation – `x`^{2} `+ 3x -15 = 0`

- … because we “fixed” the values of `a`, `b` and `c`

The primary problem we need to solve is finding out a way to take inputs “from the user”

- This will make the program more “generic”

The problems with our solver !!

First, we could only solve one specific equation – $x^2 + 3x - 15 = 0$

- … because we “fixed” the values of `a`, `b` and `c`

The primary problem we need to solve is finding out a way to take inputs “from the user”

- This will make the program more “generic”

Second, the output that the program produced looked a bit ugly

- The prompt started at the same point, where the output ended

The problems with our solver !!

First, we could only solve one specific equation – `x`^{2} `+ 3x -15 = 0`

- … because we "fixed" the values of `a`, `b` and `c`

The primary problem we need to solve is finding out a way to take inputs "from the user"

- This will make the program more "generic"

Second, the output that the program produced looked a bit ugly

- The prompt started at the same point, where the output ended

The secondary problem we should mitigate is making the prompt appear on the next line

- Actually, it will be helpful if we know this, even for other messages that the program may show

The “generic” equation solver…

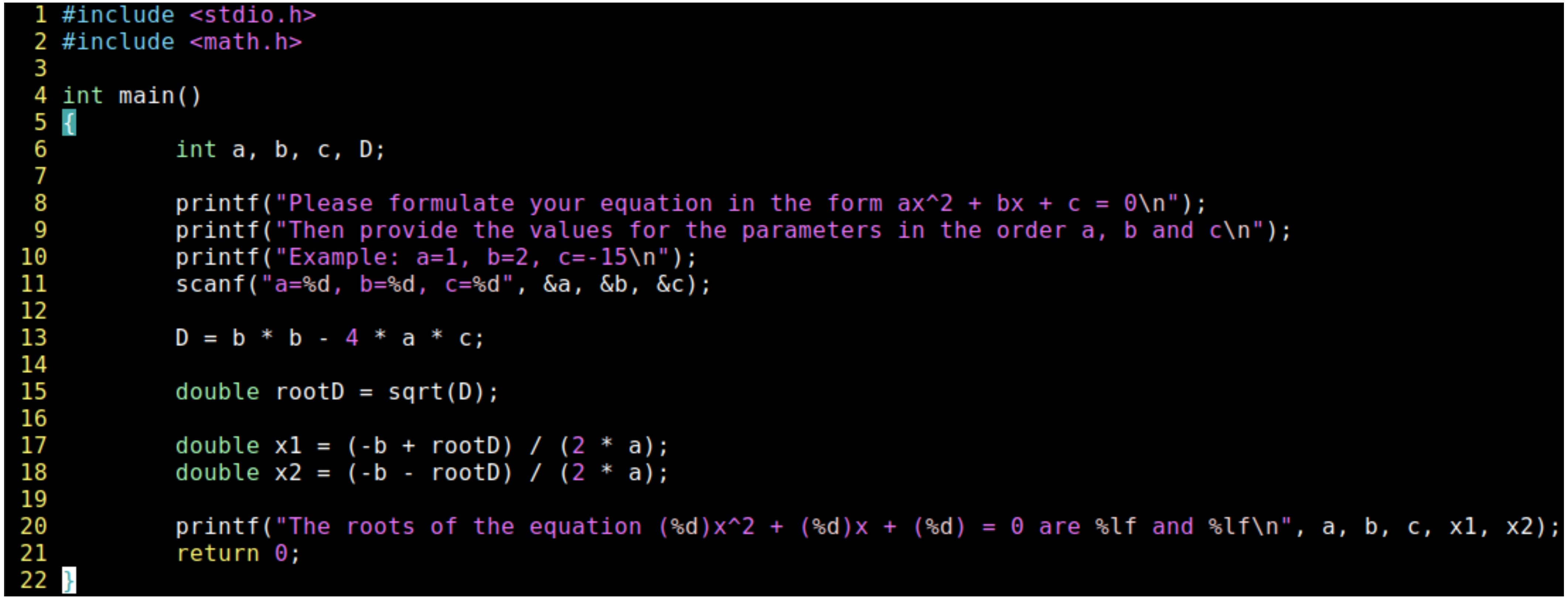
Let us create another version of our equation solver

- Let us call it `GenericQuadraticEquationWithRealRootsSolver.c`

We will make two changes here

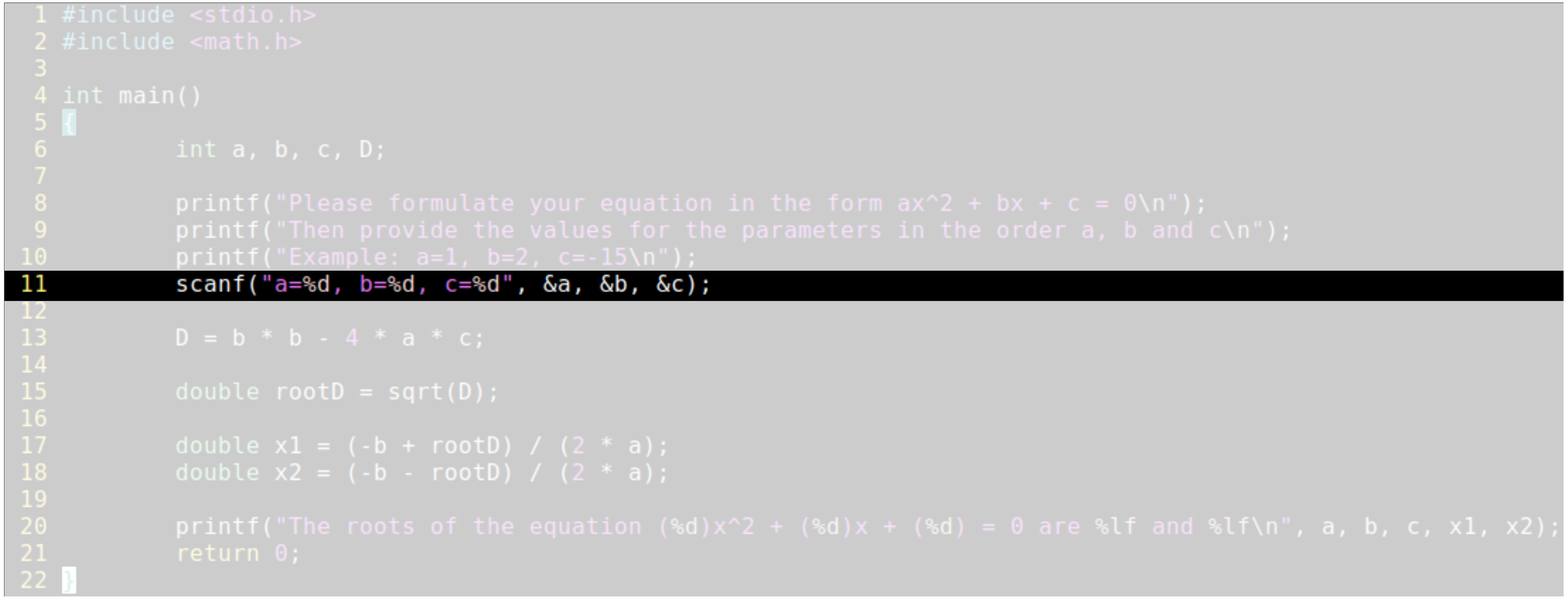
- We will use another library function, `scanf()`, to take some inputs
- We will use a “special character” in our Strings the `\n` character, which means “new line”

The “generic” equation solver…



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

int main()
{
        int a, b, c, D;

        printf("Please formulate your equation in the form ax^2 + bx + c = 0\n");
        printf("Then provide the values for the parameters in the order a, b and c\n");
        printf("Example: a=1, b=2, c=-15\n");
        scanf("a=%d, b=%d, c=%d", &a, &b, &c);

        D = b * b - 4 * a * c;

        double rootD = sqrt(D);

        double x1 = (-b + rootD) / (2 * a);
        double x2 = (-b - rootD) / (2 * a);

        printf("The roots of the equation (%d)x^2 + (%d)x + (%d) = 0 are %lf and %lf\n", a, b, c, x1, x2);
        return 0;
}
```

The “generic” equation solver…



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

int main()
{
        int a, b, c, D;

        printf("Please formulate your equation in the form ax^2 + bx + c = 0\n");
        printf("Then provide the values for the parameters in the order a, b and c\n");
        printf("Example: a=1, b=2, c=-15\n");
        scanf("a=%d, b=%d, c=%d", &a, &b, &c);

        D = b * b - 4 * a * c;

        double rootD = sqrt(D);

        double x1 = (-b + rootD) / (2 * a);
        double x2 = (-b - rootD) / (2 * a);

        printf("The roots of the equation (%d)x^2 + (%d)x + (%d) = 0 are %lf and %lf\n", a, b, c, x1, x2);
        return 0;
}
```

The syntax for `scanf()` is almost similar to `printf()`

The `scanf()` Library Function

The `printf()` function prints a message on the screen

The `scanf()` Library Function

The `printf()` function prints a message on the screen

We can use the format specifiers, along with associated variables, to create an output String

The `scanf()` Library Function

The `printf()` function prints a message on the screen

We can use the format specifiers, along with associated variables, to create an output String

The `scanf()` function works in almost the same fashion, but the role is reversed

The `scanf()` Library Function

The `printf()` function prints a message on the screen

We can use the format specifiers, along with associated variables, to create an output String

The `scanf()` function works in almost the same fashion, but the role is reversed

The `scanf()` Library Function

The `printf()` function prints a message on the screen

We can use the format specifiers, along with associated variables, to create an output String

The `scanf()` function works in almost the same fashion, but the role is reversed

Similar to `printf()`, which prints a well-formatted output, `scanf()` expects a well-formatted input

The `scanf()` Library Function

The `printf()` function prints a message on the screen

We can use the format specifiers, along with associated variables, to create an output String

The `scanf()` function works in almost the same fashion, but the role is reversed

Similar to `printf()`, which prints a well-formatted output, `scanf()` expects a well-formatted input

`scanf()` expects user to type a message as containing specific value types at specific places

The `scanf()` Library Function

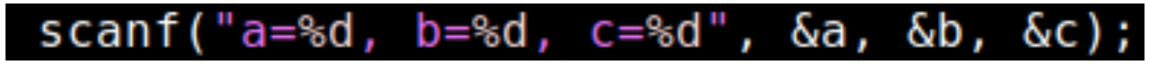
The `printf()` function prints a message on the screen

We can use the format specifiers, along with associated variables, to create an output String

The `scanf()` function works in almost the same fashion, but the role is reversed

Similar to `printf()`, which prints a well-formatted output, `scanf()` expects a well-formatted input

`scanf()` expects user to type a message as containing specific value types at specific places



```
scanf("a=%d, b=%d, c=%d", &a, &b, &c);
```

Here, `scanf()` expects the values of `a`, `b` and `c`, in exactly the way it is mentioned, e.g.

- `a=1, b=2, c=-15`

The `scanf()` Library Function

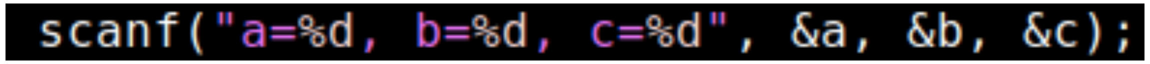
The `printf()` function prints a message on the screen

We can use the format specifiers, along with associated variables, to create an output String

The `scanf()` function works in almost the same fashion, but the role is reversed

Similar to `printf()`, which prints a well-formatted output, `scanf()` expects a well-formatted input

`scanf()` expects user to type a message as containing specific value types at specific places

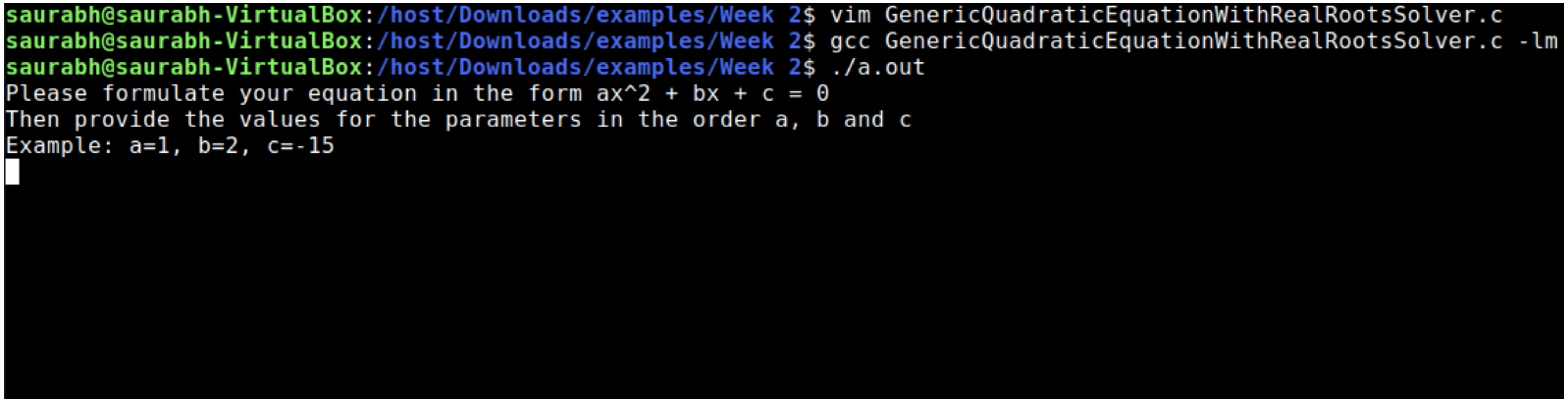


```
scanf("a=%d, b=%d, c=%d", &a, &b, &c);
```

Here, `scanf()` expects the values of `a`, `b` and `c`, in exactly the way it is mentioned, e.g.

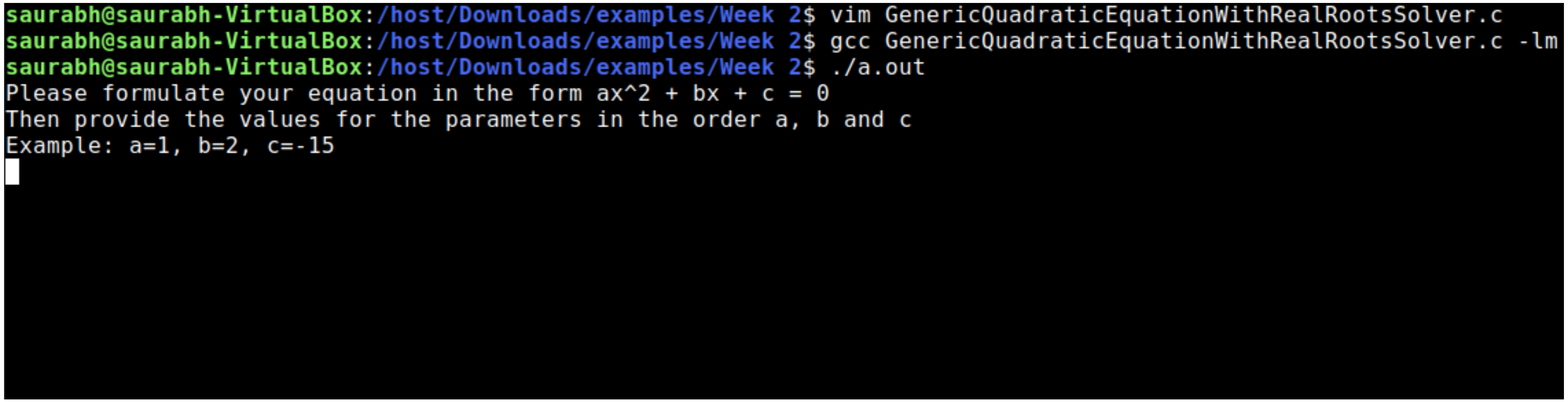
- `a=1, b=2, c=-15`
- You must add a & before all the variables – something that is different from `printf()`
- We will know what this & means in later weeks; for now, just assume this is a part of the syntax

Using the “generic” equation solver…



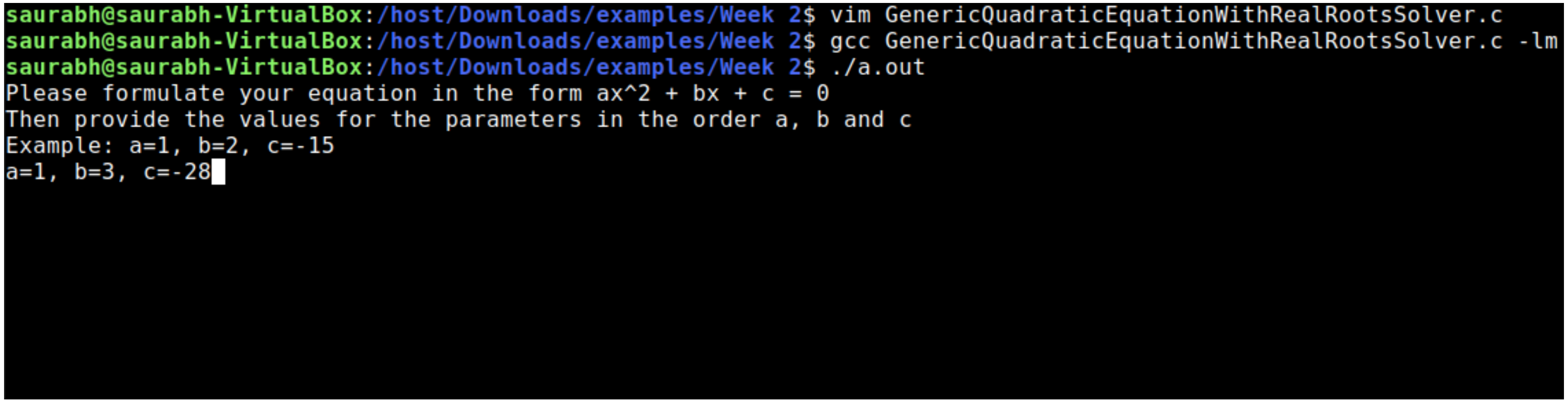
```
saurabh@saurabh-VirtualBox:/host/Downloads/examples/Week 2$ vim GenericQuadraticEquationWithRealRootsSolver.c
saurabh@saurabh-VirtualBox:/host/Downloads/examples/Week 2$ gcc GenericQuadraticEquationWithRealRootsSolver.c -lm
saurabh@saurabh-VirtualBox:/host/Downloads/examples/Week 2$ ./a.out
Please formulate your equation in the form ax^2 + bx + c = 0
Then provide the values for the parameters in the order a, b and c
Example: a=1, b=2, c=-15
```

Using the “generic” equation solver…


```
saurabh@saurabh-VirtualBox:/host/Downloads/examples/Week 2$ vim GenericQuadraticEquationWithRealRootsSolver.c
saurabh@saurabh-VirtualBox:/host/Downloads/examples/Week 2$ gcc GenericQuadraticEquationWithRealRootsSolver.c -lm
saurabh@saurabh-VirtualBox:/host/Downloads/examples/Week 2$ ./a.out
Please formulate your equation in the form ax^2 + bx + c = 0
Then provide the values for the parameters in the order a, b and c
Example: a=1, b=2, c=-15
█
```

When you run this program, your cursor will simply blink at this point – this is an indication that your program wants “inputs” from you

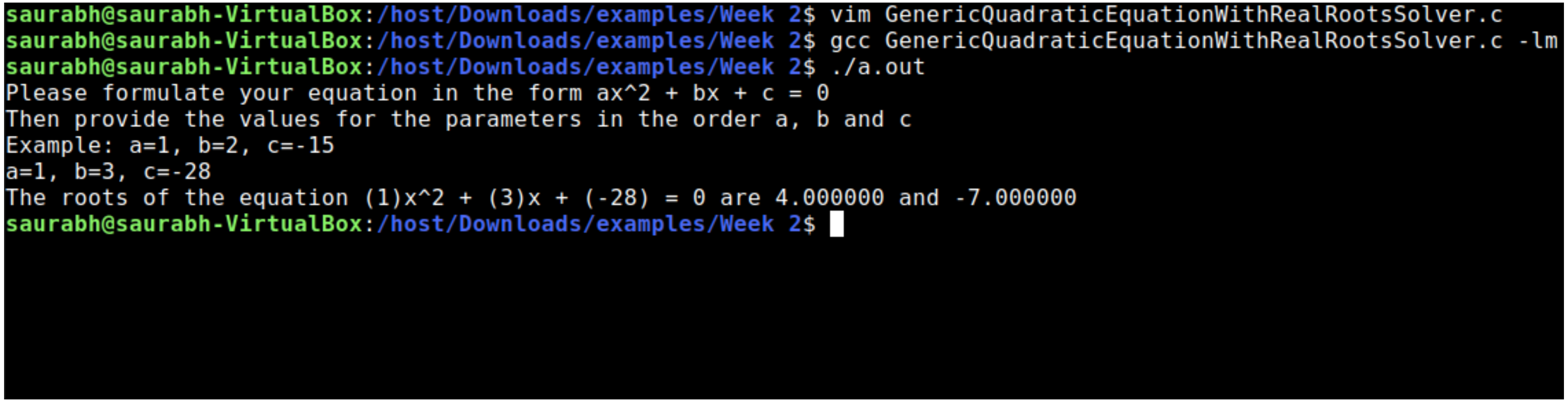
Using the “generic” equation solver…



```
saurabh@saurabh-VirtualBox:/host/Downloads/examples/Week 2$ vim GenericQuadraticEquationWithRealRootsSolver.c
saurabh@saurabh-VirtualBox:/host/Downloads/examples/Week 2$ gcc GenericQuadraticEquationWithRealRootsSolver.c -lm
saurabh@saurabh-VirtualBox:/host/Downloads/examples/Week 2$ ./a.out
Please formulate your equation in the form ax^2 + bx + c = 0
Then provide the values for the parameters in the order a, b and c
Example: a=1, b=2, c=-15
a=1, b=3, c=-28
```

Type the values of `a`, `b` and `c`, in exactly the same fashion as `scanf()` expects

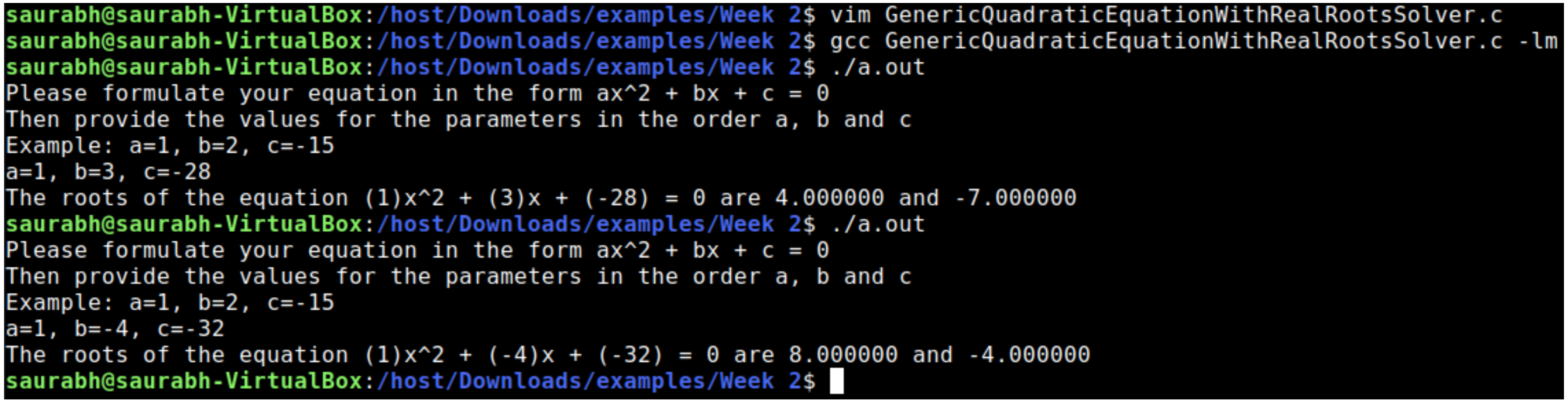
Using the "generic" equation solver…



```
saurabh@saurabh-VirtualBox:/host/Downloads/examples/Week 2$ vim GenericQuadraticEquationWithRealRootsSolver.c
saurabh@saurabh-VirtualBox:/host/Downloads/examples/Week 2$ gcc GenericQuadraticEquationWithRealRootsSolver.c -lm
saurabh@saurabh-VirtualBox:/host/Downloads/examples/Week 2$ ./a.out
Please formulate your equation in the form ax^2 + bx + c = 0
Then provide the values for the parameters in the order a, b and c
Example: a=1, b=2, c=-15
a=1, b=3, c=-28
The roots of the equation (1)x^2 + (3)x + (-28) = 0 are 4.000000 and -7.000000
saurabh@saurabh-VirtualBox:/host/Downloads/examples/Week 2$
```

… and here are your roots

Using the “generic” equation solver…



```
saurabh@saurabh-VirtualBox:/host/Downloads/examples/Week 2$ vim GenericQuadraticEquationWithRealRootsSolver.c
saurabh@saurabh-VirtualBox:/host/Downloads/examples/Week 2$ gcc GenericQuadraticEquationWithRealRootsSolver.c -lm
saurabh@saurabh-VirtualBox:/host/Downloads/examples/Week 2$ ./a.out
Please formulate your equation in the form ax^2 + bx + c = 0
Then provide the values for the parameters in the order a, b and c
Example: a=1, b=2, c=-15
a=1, b=3, c=-28
The roots of the equation (1)x^2 + (3)x + (-28) = 0 are 4.000000 and -7.000000
saurabh@saurabh-VirtualBox:/host/Downloads/examples/Week 2$ ./a.out
Please formulate your equation in the form ax^2 + bx + c = 0
Then provide the values for the parameters in the order a, b and c
Example: a=1, b=2, c=-15
a=1, b=-4, c=-32
The roots of the equation (1)x^2 + (-4)x + (-32) = 0 are 8.000000 and -4.000000
saurabh@saurabh-VirtualBox:/host/Downloads/examples/Week 2$
```

You can try it out for other combinations too !!

How did we change lines in the messages?

`printf()` can not only print well formatted Strings, it can also print them in multiple lines

How did we change lines in the messages?

`printf()` can not only print well formatted Strings, it can also print them in multiple lines

All you need is one simple character – the newline character or `\n`

How did we change lines in the messages?

`printf()` can not only print well formatted Strings, it can also print them in multiple lines

All you need is one simple character – the newline character or `\n`

Actually, `printf()` and `scanf()` use backslash (`\`) as an *escape* character

- This means that whenever they see a \ in a String, they “escape” from their normal processing
- … and check the character immediately after the backslash

How did we change lines in the messages?

`printf()` can not only print well formatted Strings, it can also print them in multiple lines

All you need is one simple character – the newline character or `\n`

Actually, `printf()` and `scanf()` use backslash (`\`) as an *escape* character

- This means that whenever they see a `\` in a String, they "escape" from their normal processing
- … and check the character immediately after the backslash

Not every character can immediately succeed a backslash, only few can

- Each one of these few, make `printf()` and `scanf()` behave differently

How did we change lines in the messages?

`printf()` can not only print well formatted Strings, it can also print them in multiple lines

All you need is one simple character – the newline character or `\n`

Actually, `printf()` and `scanf()` use backslash (`\`) as an *escape* character
- This means that whenever they see a `\` in a String, they “escape” from their normal processing
- … and check the character immediately after the backslash

Not every character can immediately succeed a backslash, only few can
- Each one of these few, make `printf()` and `scanf()` behave differently
- For example, if they find `n`, they change the current output line
- Similarly, if they find `t`, a tab space is printed (which could be 4 or 8 spaces)

How did we change lines in the messages?

`printf()` can not only print well formatted Strings, it can also print them in multiple lines

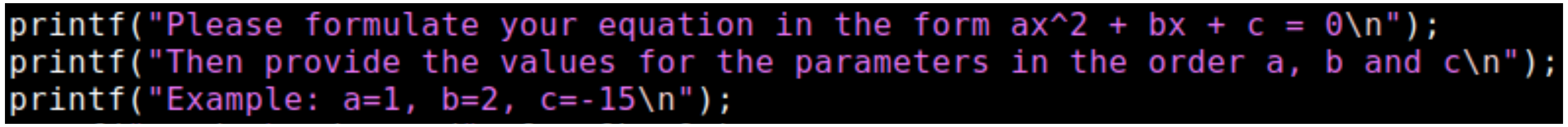
All you need is one simple character – the newline character or `\n`

Actually, `printf()` and `scanf()` use backslash (`\`) as an *escape* character

- This means that whenever they see a `\` in a String, they “escape” from their normal processing
- … and check the character immediately after the backslash

Not every character can immediately succeed a backslash, only few can

- Each one of these few, make `printf()` and `scanf()` behave differently
- For example, if they find `n`, they change the current output line
- Similarly, if they find `t`, a tab space is printed (which could be 4 or 8 spaces)



```
printf("Please formulate your equation in the form ax^2 + bx + c = 0\n");
printf("Then provide the values for the parameters in the order a, b and c\n");
printf("Example: a=1, b=2, c=-15\n");
```

How did we change lines in the messages?

`printf()` can not only print well formatted Strings, it can also print them in multiple lines

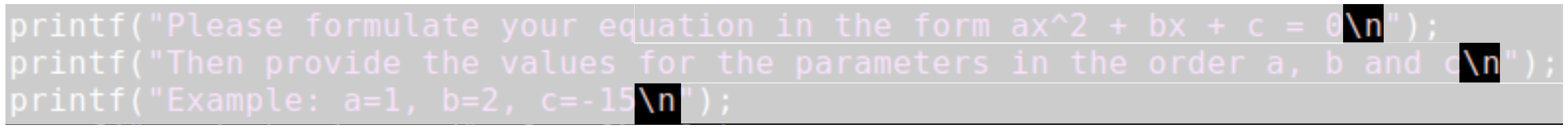
All you need is one simple character – the newline character or `\n`

Actually, `printf()` and `scanf()` use backslash (`\`) as an *escape* character

- This means that whenever they see a `\` in a String, they “escape” from their normal processing
- … and check the character immediately after the backslash

Not every character can immediately succeed a backslash, only few can

- Each one of these few, make `printf()` and `scanf()` behave differently
- For example, if they find `n`, they change the current output line
- Similarly, if they find `t`, a tab space is printed (which could be 4 or 8 spaces)



```
printf("Please formulate your equation in the form ax^2 + bx + c = 0\n");
printf("Then provide the values for the parameters in the order a, b and c\n");
printf("Example: a=1, b=2, c=-15\n");
```

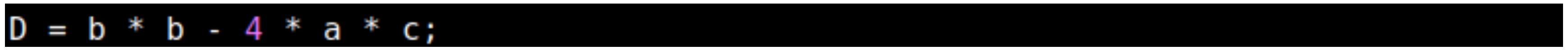
So this is how we were able to change lines in the messages we printed

Parentheses or no Parentheses

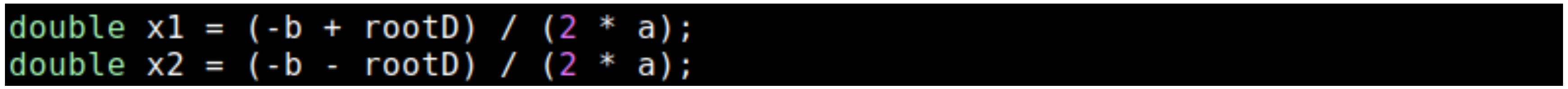
In these examples, we have done some computation

Parentheses or no Parentheses

In these examples, we have done some computation



```
D = b * b - 4 * a * c;
```

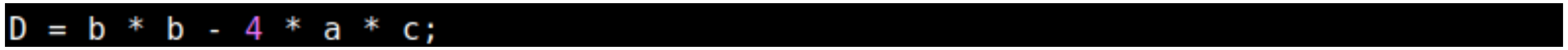


```
double x1 = (-b + rootD) / (2 * a);
double x2 = (-b - rootD) / (2 * a);
```

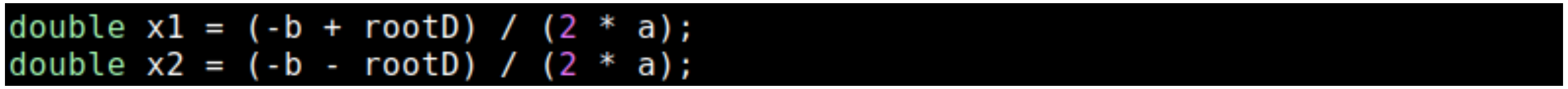
… these steps, to be precise

Parentheses or no Parentheses

In these examples, we have done some computation



```
D = b * b - 4 * a * c;
```



```
double x1 = (-b + rootD) / (2 * a);
double x2 = (-b - rootD) / (2 * a);
```

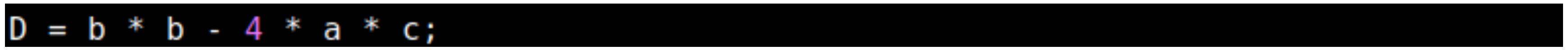
… these steps, to be precise

But something you may have noticed already: a non-uniform usage of parentheses

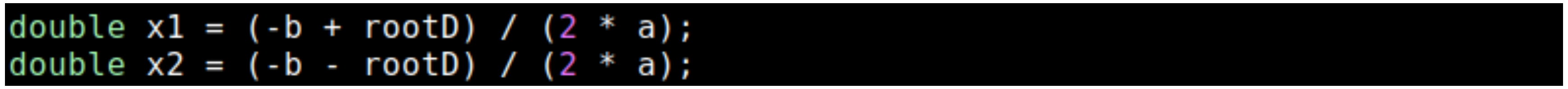
- Why are there no parentheses in the first instance, but two pairs in the second?

Parentheses or no Parentheses

In these examples, we have done some computation



```
D = b * b - 4 * a * c;
```



```
double x1 = (-b + rootD) / (2 * a);
double x2 = (-b - rootD) / (2 * a);
```

… these steps, to be precise

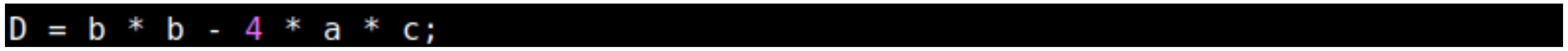
But something you may have noticed already: a non-uniform usage of parentheses

- Why are there no parentheses in the first instance, but two pairs in the second?

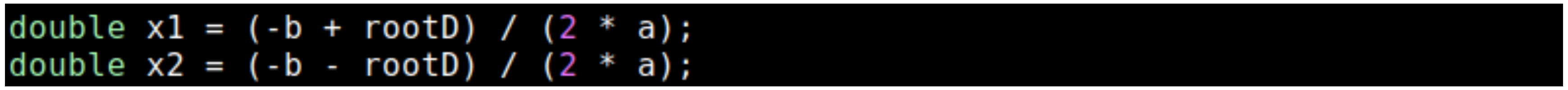
It is because we "don't need them" in the first case, but we "do need them" in the second case

Parentheses or no Parentheses

In these examples, we have done some computation



```
D = b * b - 4 * a * c;
```



```
double x1 = (-b + rootD) / (2 * a);
double x2 = (-b - rootD) / (2 * a);
```

… these steps, to be precise

But something you may have noticed already: a non-uniform usage of parentheses

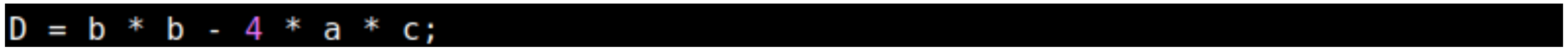
- Why are there no parentheses in the first instance, but two pairs in the second?

It is because we “don’t need them” in the first case, but we “do need them” in the second case

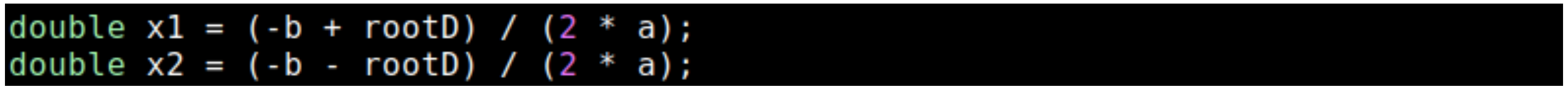
You may know something like BODMAS in arithmetic

Parentheses or no Parentheses

In these examples, we have done some computation



```
D = b * b - 4 * a * c;
```



```
double x1 = (-b + rootD) / (2 * a);
double x2 = (-b - rootD) / (2 * a);
```

… these steps, to be precise

But something you may have noticed already: a non-uniform usage of parentheses

- Why are there no parentheses in the first instance, but two pairs in the second?

It is because we “don’t need them” in the first case, but we “do need them” in the second case

You may know something like BODMAS in arithmetic

But in C, things are a bit different

Operator Precedence

In C, every *operators* has a *precedence*

- Precedence is the priority, in which the operator is evaluated, with respect to other operators

Operator Precedence

In C, every *operators* has a *precedence*

- Precedence is the priority, in which the operator is evaluated, with respect to other operators

When a C expression is evaluated, the operator with highest precedence is evaluated first

- … followed by the operator that is next in precedence, and so on…

Operator Precedence

In C, every *operators* has a *precedence*

- Precedence is the priority, in which the operator is evaluated, with respect to other operators

When a C expression is evaluated, the operator with highest precedence is evaluated first

- … followed by the operator that is next in precedence, and so on…

If two operators with the same precedence appear in an expression, one on the left is evaluated first

- … well, “almost always”… we’ll discuss the exceptions to this left to right rule some time later

Operator Precedence

In C, every *operators* has a *precedence*

- Precedence is the priority, in which the operator is evaluated, with respect to other operators

When a C expression is evaluated, the operator with highest precedence is evaluated first

- … followed by the operator that is next in precedence, and so on…

If two operators with the same precedence appear in an expression, one on the left is evaluated first

- … well, “almost always”… we’ll discuss the exceptions to this left to right rule some time later

Parentheses has the highest precedence among operators

- It has a few peers, but for now, it suffices to know that parentheses have highest precedence
- This is why parentheses can be used to finely control the evaluation sequence in an expression

Example: Evaluation of a C statement

```
D = b * b - 4 * a * c;
```

- Let us see how it is evaluated in C

Example: Evaluation of a C statement

`D = b * b - 4 * a * c;`

- Let us see how it is evaluated in C

To make it easier to explain, let us represent the `*`s in this statement as following

- `D = b` $*_1$ `b - 4` $*_2$ `a` $*_3$ `c;`
- There are a total of 5 operators in this statement: `=`, $*_1$, `-`, $*_2$, $*_3$

Example: Evaluation of a C statement

`D = b * b - 4 * a * c;`

- Let us see how it is evaluated in C

To make it easier to explain, let us represent the `*`s in this statement as following

- `D = b` $*_1$ `b - 4` $*_2$ `a` $*_3$ `c;`
- There are a total of 5 operators in this statement: $=$, $*_1$, $-$, $*_2$, $*_3$

$*$ has higher precedence than $-$, and $-$ has higher precedence than $=$ in C

Example: Evaluation of a C statement

`D = b * b - 4 * a * c;`

- Let us see how it is evaluated in C

To make it easier to explain, let us represent the $*$s in this statement as following

- `D = b` $*_1$ `b - 4` $*_2$ `a` $*_3$ `c;`
- There are a total of 5 operators in this statement: $=$, $*_1$, $-$, $*_2$, $*_3$

$*$ has higher precedence than $-$, and $-$ has higher precedence than $=$ in C

Also, since all the $*$s have the same precedence, they are evaluated from left to right

- … this means $*_1$ is evaluated first, then $*_2$, and then $*_3$

Example: Evaluation of a C statement

`D = b * b - 4 * a * c;`

- Let us see how it is evaluated in C

To make it easier to explain, let us represent the *s in this statement as following

- D = b $*_1$ b − 4 $*_2$ a $*_3$ c;
- There are a total of 5 operators in this statement: =, $*_1$, −, $*_2$, $*_3$

* has higher precedence than −, and − has higher precedence than = in C

Also, since all the *s have the same precedence, they are evaluated from left to right

- … this means $*_1$ is evaluated first, then $*_2$, and then $*_3$

Then, the − is evaluated, followed by the = at the end

Example: Evaluation of a C statement

`D = b * b - 4 * a * c;`

- Let us see how it is evaluated in C

To make it easier to explain, let us represent the `*`s in this statement as following

- `D = b` $*_1$ `b - 4` $*_2$ `a` $*_3$ `c;`
- There are a total of 5 operators in this statement: $=$, $*_1$, $-$, $*_2$, $*_3$

$*$ has higher precedence than $-$, and $-$ has higher precedence than $=$ in C

Also, since all the $*$s have the same precedence, they are evaluated from left to right

- … this means $*_1$ is evaluated first, then $*_2$, and then $*_3$

Then, the $-$ is evaluated, followed by the $=$ at the end

So the expression essentially becomes

- `D = ((b` $*_1$ `b) - ((4 * a) * c));`

Homework !!

There are a fixed set of escape characters that you can use in C; `\n` is just one of them

- Read about others
- This link gives a brief overview of `printf()`, with a short introduction to escape characters as well http://web.mit.edu/10.001/Web/Course_Notes/c_Notes/tips_printf.html

Other than the arithmetic operators, C has many others, we will explore them implicitly

- … i.e. we will use them in Programs, as and when required
- However, it may not be a bad idea to have a look at them today as well
- This link explains them in brief: https://www.geeksforgeeks.org/operator-precedence-and-associativity-in-c/
- **Don't worry much if you don't understand the concept of associativity** – we'll keep discussing it in later weeks