Diploma in

Computer Science

Operators

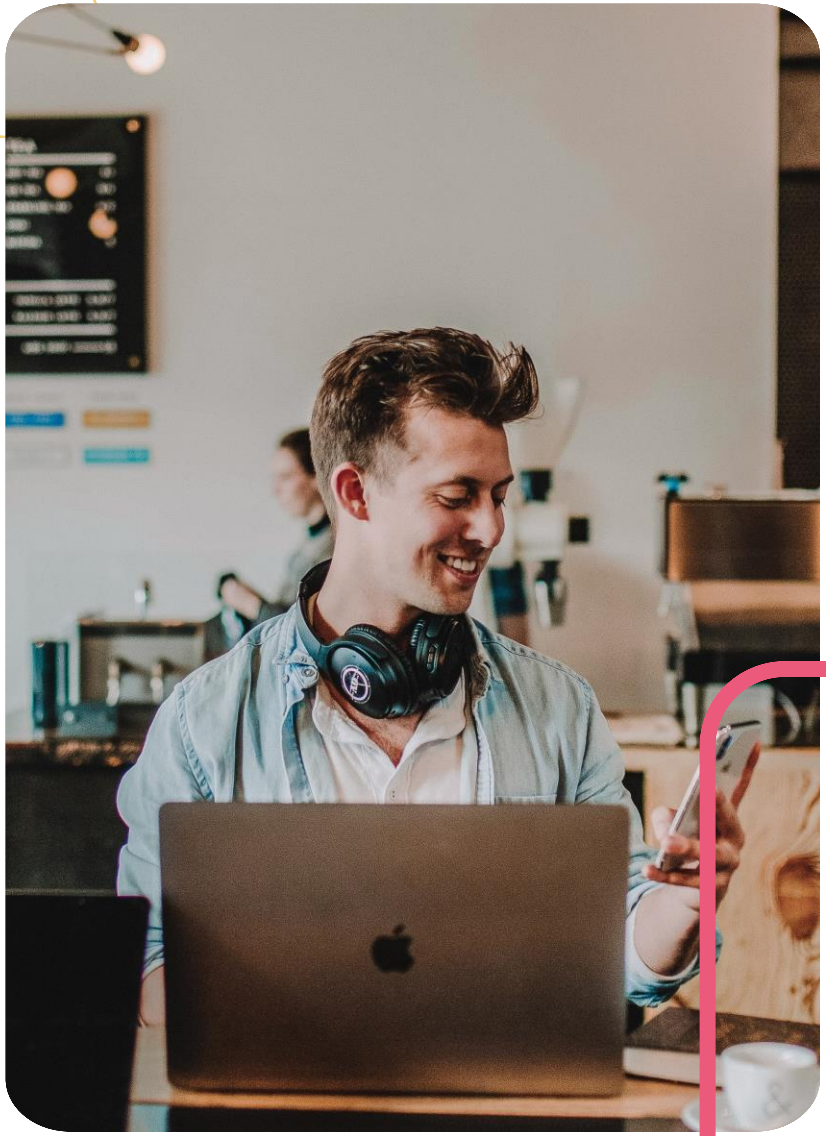
Lesson 7 Challenge Feedback

Challenge:

Ask for input from the user and perform a calculation.

Then display the result to the user.

- To accept input from the user, use the scanf function.
- Perform any calculation of your choice using the variable that held the input obtained from the user.
- Then use the printf function to display the output.



Objectives

- Define operators
- Explore increment and decrement operators in programs
- Use logic operators appropriately in code
- Discuss the various ways of assigning data to variables using operators

Arithmetic and logic operators

Addition and subtraction

+ operator for addition

– operator for subtraction

Result with variable stored on the left

```
total=2+3;
```

Example

Incrementing and decrementing

Increase the value of a variable in steps when repeating content, like this:

```
a=a+1;
```
Example

```
a++;
```
Example

```
a--;
```
Example

Pre-increment: value of variable incremented before it is used

```
#include <stdio.h>
int main()
{
        int a=0;
        while(++a < 10)
        {
            printf("%d ",a);
        }
        return 0;
}
```

Example

Output

```
1 2 3 4 5 6 7 8 9 10
```

Post-increment: variable first incremented

```
#include <stdio.h>
int main()
{
        int a=0;
        while(a++ < 10)
        {
            printf("%d ",a);
        }
        return 0;
}
```

Example

Output

```
1 2 3 4 5 6 7 8 9 10
```

Increment operators must be used with care to avoid unexpected code behaviour.

Multiplication

- Two ways: parentheses () or the asterisk *
- With parentheses, specify the multiplication operation using an asterisk because brackets mean many things in C

```
a =a*b;
a=a*(b+c);
```

Example

Division

- Use the forward slash
- Same BODMAS rule
- Result is typically not an integer
- Variable that stores results of a division operation declared as float
- Preserves the accuracy of the result

```
a=a/b;
```

Example

Logical operators

Used to execute a block of code based on a condition or a set of conditions

Operators	Example
&& (logical AND)	(a>5)&&(b<5)
|| (logical OR)	(a>=10)||(b>=10)
! (logical NOT)	((a>5)&&(b<5))

Logical operators

Useful when you want to perform an operation that depends on more than one condition

Example: automatic plant watering

Advanced operators

Relational operators

Operator	Example
>	a > b
<	a < b
>=	a >= b
<=	a <= b
==	a == b
!=	a!= b
>	a > b

They allow us to test a variable against a certain condition to see if we can go ahead and execute a block of code.

Assignment operators

Two types of assignment operators

Simple assignment operators assign a value to a variable

Compound assignment operators combine assignment = with another regular operator

Assignment operators

Operator	Example
=	a = 10;
+=	a += 10;
-=	a -= 10;
*=	a *= 10;
/=	a /= 10;.
%=	a %= 10;
^=	a ^= 10;

They work pretty much in the same way as you would solve mathematical problems on paper (except there is always an equal sign).

Compound assignment operators

Compound assignment operators shorten your code to make it more readable and efficient.

```
x=x+5;
```

```
MOVE A (X)
ADD A 5
MOVE (X) A
```

```
x+=5;
```

```
ADD (X) 5
```

DID YOU KNOW?

C is used on a lower level of abstraction than other programming languages and is typically used to build platforms on which other programs run.

Special operators

Special operators

Operator	Example
&	printf(“%d”,&a);
*****	* a, *
sizeof ()	size of (int)

Aha moment!

- To tell the computer to save the input to the address location specified in the statement:

```
Scanf(“%d”,&a);
```

Example

- In C, arguments always passed by value unless otherwise specified
- Variable ‘a’ is a copy of the actual variable
- Code snippet one entity at a time:
  scanf integer and store at address a

size of operator

- Returns the size of the variable in question

```
printf("%d”,sizeof(a));
```

Example

- Useful to dynamically allocate memory
- Allows you to get the size of the data type from the system at runtime and allocate the required space accordingly
- Can be used to calculate the number of array elements

Operator precedence

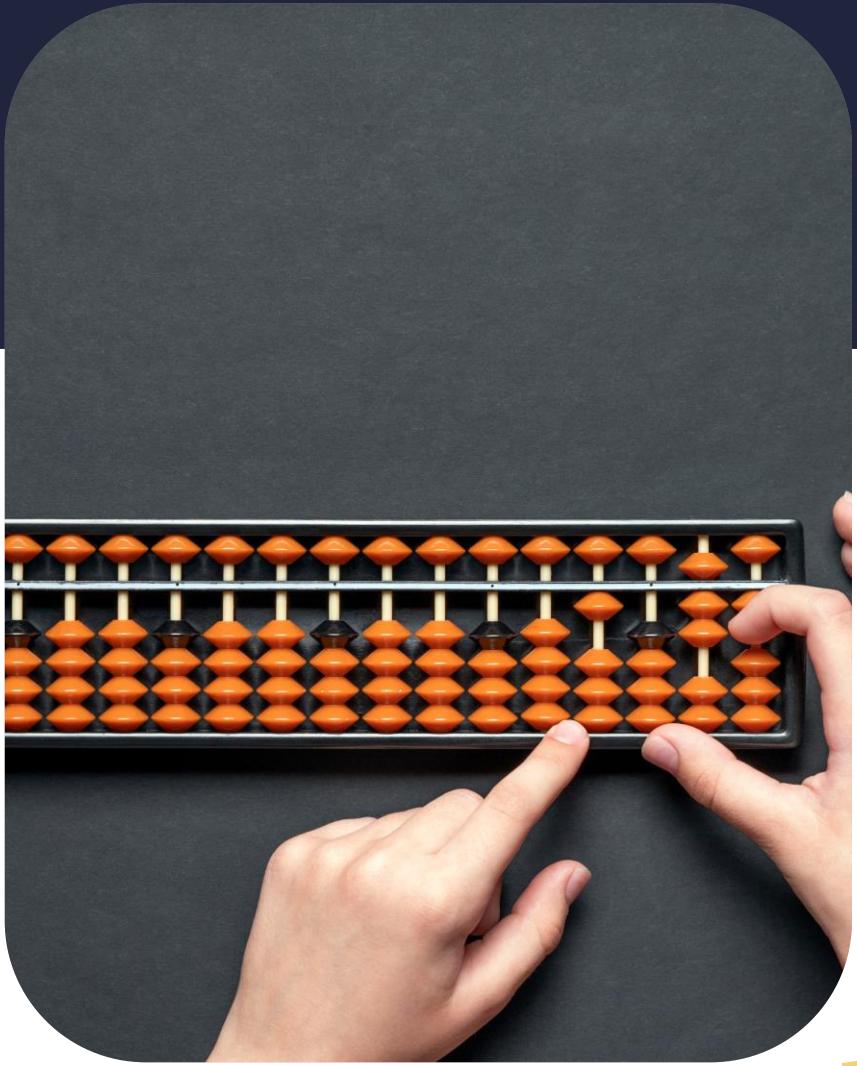
DID YOU KNOW?

BODMAS stands for:

- Brackets
- Orders/Operations
- Division
- Multiplication
- Addition
- Subtraction

Arithmetic

Operator	Operation(s)	Position
( )	Parentheses	Evaluated first
*/%	Multiplication, Division, Remainder	Evaluated second
+-	Addition, Subtraction	Evaluated third
=	Assignment	Evaluated after evaluating everything else



Classifying operators based on the number of operands they are required to complete...

- Unary
  - sizeof
  - Increment: +, ++
  - Decrement: -, --
- Binary
  - Arithmetic: +, -, *, %, /
  - Relational: ==, !=, <, >, <=, >=
  - Logical: &&, ||, !
  - Assignment: +,+=, =, *=, %=, /=, <<=, >>=, &=, ^=, |=
  - Special: &,*
- Ternary
  - Conditional: ?:

The name of the classification suggests the number of operands it requires

Unary operators require one operand to complete an operation

Binary operators require two operators

Ternary operators require 3: condition, expression for when condition is met, and expression for when condition is not met

Logical operators precedence

Operator	Precedence
!	First precedence, always evaluated first
&&	Second precedence, evaluated after the ! operator if present
||	Last precedence. Evaluated after all other logical operators have been evaluated.

Use parentheses to group operands with their operators so that they are evaluated in the correct order.

First evaluate whatever is inside parentheses.

Bitwise operations

Done at bit level

Bitwise operations are fast

Use less resources and less power

DID YOU KNOW?

According to a study by Globalwebindex 35-43% of people in the market for a new device consider battery life to be an important factor.

Operator associativity

Multiple operators at the same level of precedence

Each level of precedence has its own associativity

Why bother with precedence and associativity?

Can quite easily give you incorrect results

Example

```
a=4+5*6;
```

Apply multiplication first – result would be 34

Computer will always use the correct associativity rules no matter what you type

Computer will always do what you tell it to do

Guidelines for using operators

Precedence	Operator	Associativity
1	++ --	Left-to-right
	()	
	[]	
	.	
	->	
	(*type*){*list*}	

Guidelines for using operators

Precedence	Operator	Associativity
2	++ --	Right-to-left
	+ -	
	! ~	
	(*type*)	
	*	
	&	
	sizeof	
	_Alignof	

Guidelines for using operators

Precedence	Operator	Associativity
3	* / %	Right-to-left
4	+ -	
5	<< >>	
6	< <=	

Guidelines for using operators

Precedence	Operator	Associativity
7	== !=	Right-to-left
8	&	
9	^	
10	|	
11	&&	
12	||	

Guidelines for using operators

Precedence	Operator	Associativity
13	?:	Right-to-left
14	=	
	+= -=	
	*= /= %=	
	<<= >>=	
	&= ^= |=	
15	,	Left-to-right

The operand with the prefix ++ and __ cannot be type cast.

The operand of sizeof cannot be a typecast.

The expression in the middle of the conditional operator is parsed as though it were in parentheses.

Operands to the left of assignment operators should always be unary.

No matter what...

Parentheses

Prefix Increment/decrement

Postfix increment/decrement

Unary plus/minus

Negation/complement

Cast/address/size

BODMAS

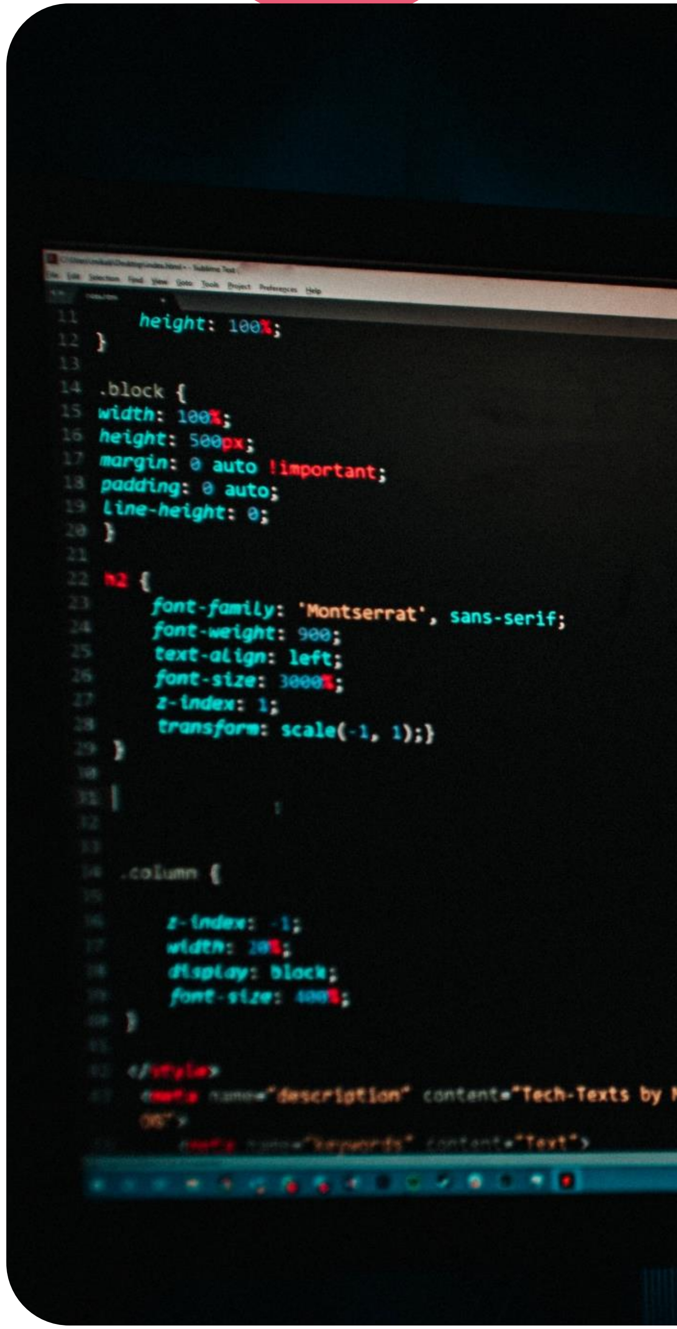
Relational

Bitwise

Ternary

Assignment

Comma



```
    height: 100%;
}

.block {
width: 100%;
height: 500px;
margin: 0 auto !important;
padding: 0 auto;
line-height: 0;
}

h2 {
    font-family: 'Montserrat', sans-serif;
    font-weight: 900;
    text-align: left;
    font-size: 3000%;
    z-index: 1;
    transform: scale(-1, 1);}
}

 .column {

    z-index: -1;
    width: 20%;
    display: block;
    font-size: 400%;
}

</style>
<meta name="description" content="Tech-Texts by
```