Definitions

Variable

In C++, a variable is a named storage location that can hold a value of a specific data type. It allows you to store and manipulate data during program execution. To define a variable in C++, you need to specify its data type and provide a name for the variable.

C++ Data Types

Basic/Primitive Data Types

The data type specifies the size and type of information the variable will store:

Data Type	Size	Description
`boolean`	1 byte	Stores true or false values
`char`	1 byte	Stores a single character/letter/number, or ASCII values
`int`	2 or 4 bytes	Stores whole numbers, without decimals
`float`	4 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 6-7 decimal digits
`double`	8 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 15 decimal digits

C++ Operators

C++ divides the operators into the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators

Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Assignment Operators

Assignment operators are used to assign values to variables =, +=, -=, *=, /=, >=, <= are examples of assignment operators

Comparison Operators

Comparison operators are used to compare two values (or variables). This is important in programming, because it helps us to find answers and make decisions.

The return value of a comparison is either `1` or `0`, which means **true** (1) or **false** (0). These values are known as **Boolean values**, and you will learn more about them in the Booleans and If..Else chapter.

==, >=, <=, !=, <, > are examples of comparison operators

Logical Operators

As with comparison operators, you can also test for **true** (`1`) or **false** (`0`) values with **logical operators**.

Logical operators are used to determine the logic between variables or values:

Operator	Name	Description
&&	Logical and	Returns true if both statements are true
||	Logical or	Returns true if one of the statements is true
!	Logical not	Reverse the result, returns false if the result is true

Unary operator

In programming, a unary operator is an operator that operates on a single operand, which can be either a variable, a literal value, or an expression. It performs an operation on the operand and produces a result.

NOT, ++, -- are examples of unary operators

Binary operator

In C++, a binary operator is an operator that operates on two operands, performing an operation between them and producing a result. The two operands can be variables, literals, or expressions. Binary operators are used to perform arithmetic, logical, bitwise, and relational operations.

+, -, *, /, AND, OR are examples of binary operators

C++ Strings

Strings are used for storing text.

A `string` variable contains a collection of characters surrounded by double quotes:

To use strings, you must include an additional header file in the source code, the `<string>` library:

String Concatenation

The `+` operator can be used between strings to add them together to make a new string. This is called **concatenation**:

Note: String is basically an array of characterschar greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'}; If you follow the rule of array initialization then you can write the above statement as follows: char greeting[] = "Hello

S.N.	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.
4	**strcmp(s1, s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
5	**strchr(s1, ch);** Returns a pointer to the first occurrence of character ch in string s1.
6	**strstr(s1, s2);** Returns a pointer to the first occurrence of string s2 in string s1.

C++ Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type, specify the name of the array followed by **square brackets** and specify the number of elements it should store:

Access the Elements of an Array

You access an array element by referring to the index number inside square brackets `[]`.

This statement accesses the value of the **first element** in **cars**:

Multi-Dimensional Arrays

A multi-dimensional array is an array of arrays.

To declare a multi-dimensional array, define the variable type, specify the name of the array followed by square brackets which specify how many elements the main array has, followed by another set of square brackets which indicates how many elements the sub-arrays have:

```
string letters[2][4];
```

Access the Elements of a Multi Dimensional Array

To access an element of a multi-dimensional array, specify an index number in each of the array's dimensions.

This statement accesses the value of the element in the **first row (0)** and **third column (2)** of the **letters** array.

```
string letters[2][4] = {
  { "A", "B", "C", "D" },
  { "E", "F", "G", "H" }
};
cout << letters[0][2]
```

C++ Conditions and If Statements

You already know that C++ supports the usual logical conditions from mathematics, You can use these conditions to perform different actions for different decisions.

C++ has the following conditional statements:

- Use `if` to specify a block of code to be executed, if a specified condition is true
- Use `else` to specify a block of code to be executed, if the same condition is false
- Use `else if` to specify a new condition to test, if the first condition is false
- Use `switch` to specify many alternative blocks of code to be executed

C++ Switch Statements

Use the `switch` statement to select one of many code blocks to be executed.

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

This is how it works:

- The `switch` expression is evaluated once
- The value of the expression is compared with the values of each `case`
- If there is a match, the associated block of code is executed
- The `break` and `default` keywords are optional, and will be described later in this chapter

The break Keyword

When C++ reaches a `break` keyword, it breaks out of the switch block.This will stop the execution of more code and case testing inside the block.When a match is found, and the job is done, it's time for a break. There is no need for more testing.

The default Keyword

The `default` keyword specifies some code to run if there is no case match:

C++ Structures

Structures (also called structs) are a way to group several related variables into one place. Each variable in the structure is known as a **member** of the structure.

Unlike an array, a structure can contain many different data types (int, string, bool, etc.).

Create a Structure

To create a structure, use the `struct` keyword and declare each of its members inside curly braces.

After the declaration, specify the name of the structure variable (**myStructure** in the example below):

```
struct {              // Structure declaration
  int myNum;          // Member (int variable)
  string myString;    // Member (string variable)
} myStructure;        // Structure variable
```

Access Structure Members

To access members of a structure, use the dot syntax (`.`):

Memory Address

In the example from the previous page, the `&` operator was used to create a reference variable. But it can also be used to get the memory address of a variable; which is the location of where the variable is stored on the computer.

When a variable is created in C++, a memory address is assigned to the variable. And when we assign a value to the variable, it is stored in this memory address.

To access it, use the & operator, and the result will represent where the variable is stored:

```
string food = "Pizza";

cout << &food;
```

Creating Pointers

A **pointer** however, is a variable that **stores the memory address as its value**.

A pointer variable points to a data type (like `int` or `string`) of the same type, and is created with the `*` operator. The address of the variable you're working with is assigned to the pointer:

```
string food = "Pizza";  // A food variable of type string
string* ptr = &food;    // A pointer variable, with the name ptr, that stores the address of food

// Output the value of food (Pizza)
cout << food << "\n";

// Output the memory address of food (0x6dfed4)
cout << &food << "\n";

// Output the memory address of food with the pointer (0x6dfed4)
cout << ptr << "\n";
```

Example explained

Create a pointer variable with the name `ptr`, that **points to** a `string` variable, by using the asterisk sign `*` (`string* ptr`). Note that the type of the pointer has to match the type of the variable you're working with.

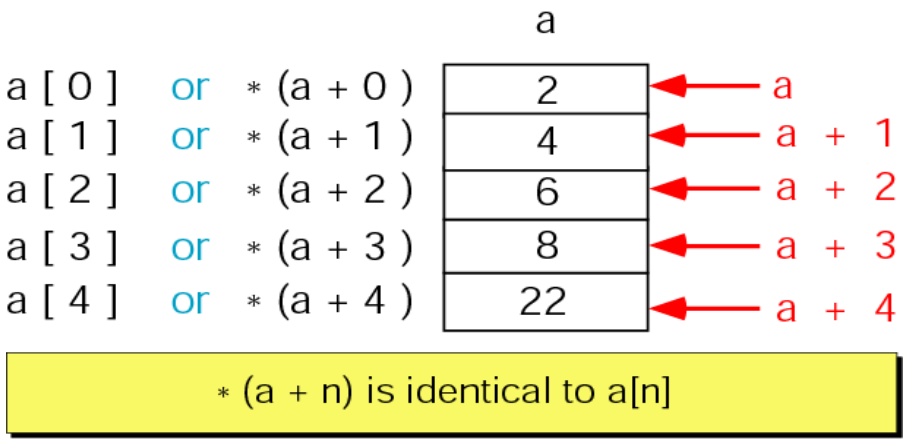
Use the `&` operator to store the memory address of the variable called `food`, and assign it to the pointer.

Now, `ptr` holds the value of `food`'s memory address.

Pointers to Arrays

The name of the array is a pointer to the first index of the array

Example→ int arr[5]; int* ptr_arr = arr;



Overloading

Function overloading is a feature in C++ that allows you to define multiple functions with the same name but different parameter lists. It enables you to create functions that perform similar tasks but can handle different types of input or different numbers of parameters. The compiler determines which function to call based on the arguments provided during the function call.

Overloading functions can have parameters of same or different data types, but the signature should be same.

Differences

If-Else and Switch Case:

1. Expression inside if statement decide whether to execute the statements inside if block or under else block. On the other hand, expression inside switch statement decide which case to execute.

2. If-else statement checks for equality as well as for logical expression. On the other hand, switch checks only for equality.

3. The if statement evaluates integer, character, pointer or floating-point type or Boolean type. On the other hand, switch statement evaluates only character or a integer datatype.

4. Sequence of execution is like either statement under if block will execute or statements under else block statement will execute. On the other hand the expression in switch statement decide which case to execute and if you do not apply a break statement after each case it will execute till the end of switch statement.

5. If expression inside if turn outs to be false, statement inside else block will be executed. If expression inside switch statement turn out to be false then default statements is executed

Break	Continue
Break statement stops the entire process of the loop.	Continue statement only stops the current iteration of the loop.
Break also terminates the remaining iterations.	Continue doesn’t terminate the next iterations; it resumes with the successive iterations.
Break statement can be used with switch statements and with loops	Continue statement can be used with loops but not switch statements.
In the break statement, the control exits from the loop.	In the continue statement, the control remains within the loop.
It is used to stop the execution of the loop at a specific condition.	It is used to skip a particular iteration of the loop.

For loop	While loop
Initialization may be either in loop statement or outside the loop.	Initialization is always outside the loop.
Once the statement(s) is executed then after increment is done.	Increment can be done before or after the execution of the statement(s).
It is normally used when the number of iterations is known.	It is normally used when the number of iterations is unknown.
Condition is a relational expression.	Condition may be expression or non-zero value.
It is used when initialization and increment is simple.	It is used for complex initialization.
For is entry controlled loop.	While is also entry controlled loop.

	Pass by Value	Pass by Reference
Parameter	A copy of the argument is passed to the function	The memory address of the argument is passed
Memory	Requires additional memory for the copy	No additional memory is required
Performance	Can be less efficient for large objects	More efficient, especially for large objects
Modifications	Modifying the parameter does not affect the original argument	Modifying the parameter affects the original argument
Return Value	Changes to the parameter are not reflected in the original argument	Changes to the parameter are reflected in the original argument
Syntax	Uses the value of the argument directly	Uses the reference (memory address) of the argument directly
Function Call	Arguments are evaluated and copied before the function call	Arguments are not copied; the reference is passed directly
Nullability	Cannot pass a null value as an argument	Can pass a null value (pointer) as an argument
Use Cases	Suitable for small objects or when modification of the argument is not desired	Suitable for large objects or when modification of the argument is required