Introduction to Computer Technology: C++ Programming Made Easier For Engineers

INTRODUCTION TO COMPUTER TECHNOLOGY

Concepts of Computer Hardware and Software

The basic concepts of computer hardware and software refer to the physical and logical components that make up a computer system.

Hardware refers to the physical components of a computer system, such as the central processing unit (CPU), memory (RAM), storage (hard drive or SSD), motherboard, power supply, and peripheral devices (such as a keyboard, mouse, and monitor).

Software, on the other hand, refers to the logical components of a computer system, such as the operating system, applications, and programming languages. An operating system (OS) is the software that controls and manages the hardware resources of a computer, and enables users to interact with the computer. Applications are programs that run on top of the operating system and perform specific tasks, such as word processing, spreadsheets, and games. Programming languages are used to write software and create applications.

Some common hardware components are:

CPU: The central processing unit is the "brain" of the computer, responsible for executing instructions and performing calculations.

RAM: Random Access Memory is a type of memory that stores data temporarily, allowing the CPU to quickly access and use it.

Storage: This refers to the devices that store data permanently, such as hard drives or solid-state drives.

Input and Output Devices: These are devices that allow users to interact with the computer, such as a keyboard, mouse, or monitor.

Some common software components are:

- **Operating Systems:** Examples include Windows, MacOS, and Linux.
- **Applications:** Examples include Microsoft Office, Adobe Photoshop, and Google Chrome.
- **Programming Languages:** Examples include Python, Java, C++ and JavaScript

Hardware and software are the two main components of a computer system, and they work together to perform various tasks and operations. Hardware refers to the physical components of a computer, while software refers to the logical components that run on top of the hardware.

Components of Computer Hardware

The basic components of computer hardware include the following:

1. **Central Processing Unit (CPU):** The brain of the computer, responsible for processing data and executing instructions.
2. **Memory:** The storage area where data and instructions are temporarily stored while the CPU processes them. There are two types of memory: Random Access Memory (RAM) and Read-Only Memory (ROM).
3. **Storage:** The area where data and instructions are permanently stored. Examples include hard disk drives (HDD) and solid-state drives (SSD).
4. **Motherboard:** The main circuit board of the computer that connects all the internal components.
5. **Power supply:** Converts and regulates the electrical power to provide the necessary power to the internal components.
6. **Input devices:** Allow the user to input data and instructions into the computer. Examples include keyboard, mouse, and scanner.
7. **Output devices:** Allow the computer to display or output data and results. Examples include monitor, printer, and speakers.
8. **Expansion cards:** Additional components that can be added to the computer to expand its capabilities. Examples include graphics card, sound card, and network card.
9. **Peripherals:** Additional devices that can be connected to the computer to expand its capabilities. Examples include external hard drives, printers, and cameras.

All these components work together to allow the computer to process data and instructions and produce output. The CPU retrieves instructions and data from memory, performs calculations, and stores the results back in memory. The storage devices store the data and instructions permanently, while input and output devices allow for interaction with the user. The motherboard

connects all the internal components and controls their communication, and the power supply provides power to all the components.

Types of Computer Systems and their Uses

Each type of computer system is designed to meet the specific needs and requirements of the tasks they are intended to perform. Personal computers are designed for general-purpose use, while workstations and servers are designed for specific tasks requiring more power and resources. Mainframes and supercomputers are used for large-scale data processing and complex simulations, embedded systems are specialized for a specific purpose, mobile devices are portable and gaming consoles are designed for gaming.

There are several types of computer systems, each with their own specific use and application:

- **Personal Computer (PC):** A computer designed for personal use, typically used for tasks such as word processing, internet browsing, and gaming.
- **Workstation:** A high-performance computer designed for tasks that require more power and resources than a typical personal computer, such as video editing, 3D rendering, and scientific research.
- **Server:** A computer designed to provide services and resources to other computers over a network. Examples include web servers, email servers, and database servers.
- **Mainframe:** A large and powerful computer designed for use by large organizations and government agencies, typically used for tasks such as banking, airline reservation systems, and government record keeping.
- **Supercomputer:** The most powerful type of computer, designed for tasks that require massive amounts of computational power, such as weather forecasting, scientific research, and nuclear simulations.
- **Embedded systems:** Small specialized computer systems that are embedded into other devices or products, such as consumer electronics, automobiles, and industrial control systems.
- **Mobile devices:** Portable computer systems, such as smartphones and tablets, that are designed for use on the go.
- **Gaming console:** A specialized computer system designed for playing video games, typically connected to a TV or monitor.

Types of Software and Their Functions

There are several types of software, each with its own specific function and application:

- **System software:** Software that controls the basic functions of a computer and its components, such as the operating system, device drivers, and utilities. Examples include Windows, Mac OS, and Linux.
- **Application software:** Software that is designed to perform specific tasks or functions, such as word processing, spreadsheet, and email. Examples include Microsoft Office, Google Docs, and Adobe Photoshop.
- **Utility software:** Specialized software that helps the user to maintain and optimize the computer system, such as antivirus, backup, and disk defragmenter.
- **Game software:** Software that is designed to provide interactive entertainment, such as video games. Examples include Super Mario, FIFA, and Call of Duty.
- **Web-based software:** Software that is accessed and used through a web browser, typically hosted on a remote server. Examples include Google Drive, Salesforce, and Trello.
- **Mobile apps:** Software that is designed for use on mobile devices, such as smartphones and tablets. Examples include WhatsApp, Instagram, and Uber.
- **Embedded software:** Software that is designed to run on embedded systems, such as consumer electronics, automobiles, and industrial control systems.
- **Artificial Intelligence (AI) software:** Software that is designed to simulate human intelligence and decision-making, such as chatbots, self-driving cars, and speech recognition.

Remember each type of software is designed to perform specific tasks, System software provides the basic functionality of the computer, application software performs specific tasks, utility software helps to maintain and optimize the computer, game software provides interactive entertainment, web-based software is accessed through a web browser, mobile apps are designed for mobile devices, embedded software is designed for embedded systems and AI software simulates human intelligence and decision-making.

Concepts and Principles of Programming

These concepts and principles listed below are fundamental to programming and are used in all programming languages. They are the building blocks that are used to create software and applications, and understanding them is essential for creating efficient and effective code.
The basic concepts and principles of programming include:

- **Algorithm:** A set of instructions that are followed to accomplish a specific task or solve a problem.
- **Syntax:** The rules and structure of a programming language, including keywords, operators, and statements.
- **Variables:** A named location in memory that can be used to store data and values.
- **Data Types:** The type of data that a variable can store, such as integers, strings, and boolean values.
- **Conditional Statements:** Instructions that determine whether a block of code should be executed based on certain conditions.
- **Loops:** Instructions that repeat a block of code a specific number of times or until a certain condition is met.
- **Functions:** A block of code that can be called and executed multiple times with different inputs.
- **Arrays:** A data structure that stores a collection of data of the same type.
- **Objects:** A data structure that stores a collection of data and functions that operate on that data.
- **Debugging:** The process of identifying and correcting errors in the code.
- **Comments:** Explanatory notes added to the code to make it more readable and understandable.
- **Modularity:** Breaking a program down into smaller, manageable pieces that can be developed, tested, and maintained separately.
- **Abstraction:** Hiding the underlying complexity of a system or process and only exposing the necessary information to the user.
- **Reusability:** The ability of a piece of code to be used in multiple places or projects.
- **Efficiency:** Using the least number of resources to achieve the desired outcome.

Introduction to C++ Programming

Programming is the process of designing and implementing a set of instructions for a computer to follow in order to perform a specific computing task or solve a problem. It involves writing and testing code, as well as debugging and maintaining the program. C++ is a popular programming language that was created in 1979 by Bjarne Stroustrup. It is an extension of the C programming language and is commonly used in the development of computer software, games, and operating systems.

C++ is a high-level language, which means that it is closer to human language than machine language. This makes it easier to read and understand, but it also requires a compiler to translate the code into machine language that a computer can understand. It is known for its versatility, efficiency, and object-oriented features. It is used in a wide range of industries, including finance, scientific research, and video game development.

To become proficient in C++, it is important to have a strong foundation in computer science concepts such as algorithms, data structures, and object-oriented programming. It is also important to have a good understanding of the C++ language syntax and be able to use various libraries and frameworks to develop solutions to complex problems.

Advantages of C++

1. **Object-oriented programming:** C++ allows for the creation and use of classes, which enables developers to create reusable and modular code.
2. **High performance:** C++ is a compiled language, which means that it is converted into machine code before execution. This allows for faster execution times compared to interpreted languages.
3. **Versatility:** C++ can be used to develop a wide range of applications, including operating systems, web browsers, games, and more.
4. **Rich library support:** C++ has a large standard library and a wide range of third-party libraries available, which makes it easier to develop complex applications.
5. **Community support:** C++ has a large and active community of developers, which makes it easier to find help and resources when needed.

6. **Compatible with other languages:** C++ can easily be integrated with other languages, such as C, making it easier to incorporate existing code into a C++ project.

ALGORITHM

Algorithms are a set of instructions or steps that are followed in a specific order to solve a problem or accomplish a task. They are used in computer science and other fields to perform a variety of tasks, such as sorting data, searching for specific information, and analyzing patterns. Algorithms can be designed using various programming languages and can be applied to a wide range of applications, including data analysis, machine learning, and artificial intelligence.

There are numerous algorithms that can be implemented in C++ programming. Some examples include:

Sorting algorithms: These algorithms are used to arrange a list of elements in a particular order, such as ascending or descending. Examples include bubble sort, insertion sort, and merge sort.

Search algorithms: These algorithms are used to search for a specific element within a list of elements. Examples include linear search, binary search, and depth-first search.

Graph algorithms: These algorithms are used to solve problems involving graphs, such as finding the shortest path between two nodes or determining whether a graph is connected. Examples include Dijkstra's algorithm and the Floyd-Warshall algorithm.

String algorithms: These algorithms are used to manipulate strings, such as searching for a specific substring within a larger string or finding the longest common subsequence between two strings. Examples include the Knuth-Morris-Pratt algorithm and the Levenshtein distance algorithm.

Computational geometry algorithms: These algorithms are used to solve problems involving geometric shapes, such as finding the intersection point of two lines or determining the convex hull of a set of points. Examples include the Bentley-Ottmann algorithm and the Graham scan algorithm.

How To Setup C++ Development Environment

1. **Install a C++ compiler:** In order to develop C++ programs, you will need a C++ compiler installed on your computer. Some popular options include GCC (GNU Compiler Collection), Clang, and Microsoft Visual C++. You can typically find these compilers included with your operating system or available for download from their respective websites.
2. **Choose an integrated development environment (IDE):** An IDE is a software application that provides a comprehensive development environment for C++ programming. Some popular IDEs for C++ include Microsoft Visual Studio, Eclipse, and Code::Blocks. These IDEs typically include features such as code completion, debugging tools, and syntax highlighting.
3. **Set up a text editor:** If you prefer to work with a text editor rather than an IDE, you will need to choose a text editor that is suitable for C++ programming. Some popular options include Sublime Text, Notepad++, and Atom. These text editors typically include syntax highlighting and basic code completion features.
4. **Configure your build settings**: Depending on your compiler and IDE/text editor, you will need to configure your build settings to specify how your code should be compiled and run. This may include setting up build paths, linking to libraries, and specifying any command line arguments.
5. **Test your setup:** Once you have your compiler, IDE/text editor, and build settings configured, you can test your setup by writing and compiling a simple C++ program. This will ensure that everything is working properly and that you are ready to begin C++ development.

Run C++ in Visual Studio Code

To run C++ in Visual Studio Code, you will need to have a C++ compiler installed on your system. If you don't already have a C++ compiler installed, you can download and install one from the internet. Some popular C++ compilers include GCC, Clang, and Microsoft Visual C++.

Use this link to download and install a compiler.
http://bit.ly/mingw10
Once you have a C++ compiler installed, follow these steps to install Visual Studio Code on Windows.

1. Go to the Visual Studio Code website (https://code.visualstudio.com/) and click the "Download" button.
2. Run the installer that downloads.
3. Follow the prompts to install Visual Studio Code on your system.

Support video link:
https://www.youtube.com/watch?v=jvg4VtYEhKU

Basic Syntax in C++

In programming, syntax refers to the set of rules that govern the structure of a program. In C++, the syntax specifies the rules for naming functions, the correct order of keywords, constants, variables, and rules for writing statements and expressions.

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5  cout << "Hello World!";
6 return 0;
7 }
```

Example explained

Line 1: #include <iostream> is a header file library that lets us work with input and output objects, such as cout (used in line 5). Header files add functionality to C++ programs.

Line 2: using namespace std means that we can use names for objects and variables from the standard library.

Line 3: A blank line. C++ ignores white space. But we use it to make the code more readable.

Line 4: Another thing that always appears in a C++ program, is int main(). This is called a function. Any code inside its curly brackets {} will be executed.

Line 5: cout (pronounced "see-out") is an object used together with the *insertion operator* (<<) to output/print text. In our example it will output "Hello World".

Line 6: return 0 ends the main function.

Alternatively

You can also rewrite the code for without importing **using namespace std**
and replacing with the std keyword, followed by the :: operator for some objects:

```
#include <iostream>

int main(){
        Std::cout  << "Hello World";
   // prints out Hello World

    return 0;
}
```

Code

```
int x;
float y = 3.14;
```

Function declarations:

```
int add(int x, int y) {
  return x + y;
}
```

If statements:

```
if (x < y) {
  std::cout << "x is less than y" << std::endl;
}
```

Loops:
Code

```
for (int i = 0; i < 10; i++) {
  std::cout << i << std::endl;
}
```

Here is a simple example of a C++ program that prints the message "Hello, World!" to the console:

```
#include <iostream>

int main()
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

This program consists of the following elements:

The **#include** directive tells the preprocessor to include a header file called iostream, which provides input and output functions.

The i**nt main()** function is the entry point of the program. It is the function that is called when the program starts.

The **std::cout** object is used to output text to the console. It is followed by the **<<** operator, which is used to send data to **std::cout.**

The **std::endl** object is used to insert a newline character after the message.

OUTPUT IN C++

To output in C++, you can use the **cout** function. Here's an example:

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

This will print the string **"Hello, world!"** followed by a newline to the standard output stream.

<std::endl; at the end creates a new line at the end of string **"Hello World"**

You can also use the **printf** function from the stdio library for output, like this:

```
#include <cstdio>

int main()
{
    printf("Hello, world!\n");
    //print out Hello, World  with a new line
    return 0;
}
```

Both of these examples will produce the same output with a New line

INPUT IN C++

There are several ways to read input in C++:

cin: This is a stream object that can be used to read input from the standard input (usually the keyboard).

cin is a predefined variable that reads data from the keyboard with the extraction operator (>>).

```
#include <iostream>
using namespace std;

int main() {
    int x;
    cout << "Enter a number: ";
    cin >> x;
    cout << "You entered: " << x << endl;
    return 0;
}
```

getline: This function can be used to read a line of text from the standard input. For example:

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string line;
    cout << "Enter a line of text: ";
    getline(cin, line);
    cout << "You entered: " << line << endl;
    return 0;
}
```

fgets: This function can be used to read a line of text from a file or from the standard input.

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

int main() {
    char line[100];
```

```
    printf("Enter a line of text: ");
    fgets(line, sizeof(line), stdin);
    printf("You entered: %s", line);
    return 0;
}
```

scanf: This function can be used to read formatted input from the standard input. For example:

```
#include <stdio.h>

int main() {
    int x;
    printf("Enter a number: ");
    scanf("%d", &x);
    printf("You entered: %d\n", x);
    return 0;
}
```

fscanf: This function can be used to read formatted input from a file. It works in a similar way to scanf.

C++ COMMENT SYNTAX

To add a comment in C++, you can use the // symbol. Anything on the same line following // will be treated as a comment by the compiler and will not be executed.

For example:

```
int main() {
  // This is a comment
  std::cout << "Hello, world!" << std::endl;
  return 0;
}
```

You can also use /* and */ to create a block comment, which can span multiple lines. Anything between /* and */ will be treated as a comment and will not be executed.

For example:

```
int main() {
  /* This is a
  multi-line comment */
  std::cout << "Hello, world!" << std::endl;
  return 0;
}
```

Variables

These are containers for storing data values. Each variable has a name and a data type (such as string, integer, or boolean). A variable in C++ is a named location in memory where a programmer can store a value. It can be a number, a string, or a boolean value (true or false).

To declare a variable in C++, you must specify the type of the variable (such as int, float, char, etc.), followed by the name of the variable. For example:

int age;
float height;
char grade;

You can also initialize a variable at the same time as you declare it by assigning it a value:

```
int age = 20;
```

```
float height = 5.6;
```

```
char grade = 'A';
```

You can also declare multiple variables of the same type by separating them with commas:

```
int age1 = 20, age2 = 30;
float height1 = 5.6, height2 = 6.1;
char grade1 = 'A', grade2 = 'B';
```

It is important to note that variables in C++ are case-sensitive, meaning that the variables "age" and "Age" are considered to be different variables. It is also important to choose descriptive and meaningful names for your variables, as this makes your code easier to understand and maintain.

Continue Statement

Continue Statement: This control structure allows the programmer to skip the remainder of the current iteration of a loop and move on to the next iteration.
The continue statement in C++ is used to skip the current iteration of a loop and move on to the next iteration. It can be used in while, do-while, and for loops. When the continue statement is encountered in a loop, the loop will immediately skip the rest of the current iteration and move on to the next iteration.

For example:

```
for (int i = 0; i < 10; i++) {
  if (i % 2 == 0) {
    continue;  // skip this iteration if i is even
  }
  cout << i << endl;
}
```

This will print the odd numbers from 1 to 9. The continue statement causes the loop to skip over the cout statement when i is even, and move on to the next iteration.

```
#include <iostream>
using namespace std;

int main() {
int num1, num2;
int result;

cout << "Enter two integers: ";
cin >> num1 >> num2;

result = num1 + num2;
cout << "The sum of the two numbers is: " << result << endl;

result = num1 - num2;
cout << "The difference of the two numbers is: " << result << endl;
```

```
result = num1 * num2;
cout << "The product of the two numbers is: " << result << endl;

result = num1 / num2;
cout << "The quotient of the two numbers is: " << result << endl;

result = num1 % num2;
cout << "The remainder of the two numbers is: " << result << endl;

return 0;
}
```

DATA TYPES IN C++

Data types: These specify the kind of data that a variable can hold. There are several data types in C++, including integers (whole numbers), floating point numbers (numbers with a decimal point), and strings (text).

Integer

An integer in C++ is a data type that stores a whole number value. It can be either a signed or unsigned integer, depending on whether it can hold negative or positive values only.
The size of an integer in C++ depends on the compiler and the system it is being compiled on. The most common sizes are 16 bits (short int), 32 bits (int), and 64 bits (long int).

Examples of declaring and initializing integers in C++:

int x = 5; // Declares an integer x and initializes it with the value 5
short int y = -2; // Declares a short integer y and initializes it with the value -2
unsigned int z = 10; // Declares an unsigned integer z and initializes it with the value 10

Integer variables can be used in arithmetic operations, such as addition, subtraction, multiplication, and division.

Example of using integers in arithmetic operations:

```
int a = 5, b = 3, c;
c = a + b; // c is now 8
c = a - b; // c is now 2
c = a * b; // c is now 15
c = a / b; // c is now 1 (integer division)
```

Float

Floating Point data type for numbers with decimal points, such as 3.14 or 0.5. In C++, a floating-point number is a numerical data type that can store decimal values. It is a type of real number, which means it can represent values with a fractional component. Floating point numbers are stored in a specific format called "floating point representation," which allows them to represent a wide range of values with a limited number of bits.

There are two main types of floating-point numbers in C++:

- single precision (float) and double precision (double).

- Single precision floating point numbers use 32 bits to store a value, while double precision floating point numbers use 64 bits.

Floating point numbers are generally used for scientific and technical calculations, where precision is important. They are also used for financial calculations, where decimal values need to be stored and manipulated.
However, floating point numbers have limited precision, which means they can only represent a certain number of decimal places accurately. This can cause issues when working with very large or very small values, or when performing calculations with a high degree of precision. In these cases, it may be necessary to use a different data type, such as long double or decimal.

Boolean

Boolean is a data type that can hold only two values: true or false. Boolean in C++ is a data type that can hold only two values: true or false. It is used to represent a logical value or condition.

To use boolean in C++, you need to include the header file "stdbool.h" and then you can declare a boolean variable like this:

bool myBoolean = true;

You can also assign a boolean value to a variable using the keywords "true" or "false".
You can also use boolean in conditional statements such as if and while loops. For example:

```
if (myBoolean)
{
// code to execute if myBoolean is true
}
else
{
// code to execute if myBoolean is false
}
```

You can also use boolean operators such as && (AND), || (OR), and ! (NOT) to compare boolean values. For example:

```
if (myBoolean && otherBoolean)
{
// code to execute if both myBoolean and otherBoolean are true
}

if (myBoolean || otherBoolean)
{
// code to execute if either myBoolean or otherBoolean are true
}

if (!myBoolean)
{
// code to execute if myBoolean is false
}
```

Character

Character is a data type for storing a single character, such as 'a' or 'Z'. Character in C++ is a data type that represents a single character in a computer's memory. It is represented by the char keyword and can hold any single ASCII or Unicode character.

For example:

```
char myChar = 'a'; // represents the character 'a'
char myChar2 = '@'; // represents the character '@'
char myChar3 = ' '; // represents the space character
```

Characters in C++ can be used in various ways, such as for input and output, for storing strings, and for storing character data in arrays. They can also be compared and manipulated using various operators, such as the equality operator (==) and the concatenation operator (+).

String

String is a data type for storing a sequence of characters, such as "**hello**" or "**goodbye**". A string in C++ is a sequence of characters stored in a contiguous block of memory. It is typically stored as an array of characters and is terminated with a null character ('\0').

It is a part of the standard library and is implemented as a class template called **std::basic_string.**
Example of how you can use strings in C++:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string greet; // greet is an empty string
  string name = "John"; // name is an assigned to John
  string greeting = "Good Afternoon!";
// greeting is a string initialized with the given value Hello!

  cout << "name: " << name << std::endl;
  cout << "greeting: " << greeting << endl;

  greet = greeting + " " + name;
// concatenate name and greeting, and store the result in greets

  cout << "greets: " << greet << endl;
  return 0;
```

Output:

```
name: John
greeting: Good Afternoon!
greets: Good Afternoon! John
```

There are many other operations that you can perform on strings, such as finding the length of a string, accessing individual characters, comparing strings, etc. You can find more information about strings in the C++ standard library documentation.

In C++, there are two main types of strings:

1. **std::string:** This is the most commonly used string type in C++. It is part of the C++ standard library and is implemented as a class template called std::basic_string. It provides a lot of useful functions for manipulating strings, such as finding the length of a string, concatenating strings, comparing strings, etc.

2. **char*:** This is a null-terminated character array, also known as a C-style string. It is not part of the C++ standard library and is implemented as an array of characters that ends with a special character called the null character ('\0'). C-style strings are generally less efficient and less flexible than std::string, but they are sometimes used for compatibility with older C code or for certain low-level operations.

To create a string in C++, you can use the following syntax:

Import the <string> library and using **namespace std without using std::**

```
string greet = “Good Morning”;
```

To create a string in C++, you can use the following syntax:

```
std::string myString = "Hello, World!";
```

You can also create a string using the string class constructor:

```
std::string myString(5, 'a'); // creates a string with 5 'a's
```

You can also initialize a string using an array of characters:

```
char myCharArray[] = {'H', 'e', 'l', 'l', 'o', '\0'};
std::string myString(myCharArray);
```

There are various functions and methods available for manipulating strings in C++, such as concatenation, finding substrings, replacing characters, etc.
Some examples of string manipulation in C++ are:

```
// concatenation
std::string str1 = "Hello";
std::string str2 = "World";
std::string str3 = str1 + " " + str2; // str3 is "Hello World"
```

```
// find substring
std::string str = "Hello World";
int pos = str.find("World"); // pos is 6

// replace character
std::string str = "Hello World";
str.replace(5, 1, ','); // str is "Hello, World"
```

It is important to note that strings in C++ are immutable, meaning that once they are created, they cannot be modified. Any changes to a string will result in the creation of a new string object.

Here is an example of how you can use both types of strings in C++:

```
#include <iostream>
#include <string>

int main()
{
  // Declare an std::string object
  std::string str1 = "Hello, world!";

  // Declare a char* and initialize it with the same value as str1
  char* str2 = "Hello, world!";

  // Print the contents of both strings
  std::cout << "str1: " << str1 << std::endl;
  std::cout << "str2: " << str2 << std::endl;

  // Find the length of str1 using the size() function
  std::cout << "Length of str1: " << str1.size() << std::endl;

  // Find the length of str2 using a loop
  int length = 0;
  for (; str2[length] != '\0'; ++length);
  std::cout << "Length of str2: " << length << std::endl;

  return 0;
}
```

Output:

```
str1: Hello, world!
str2: Hello, world!
Length of str1: 13
Length of str2: 13
```

Array: a data type that allows you to store multiple values in a single variable. An array in C++ is a collection of variables that are stored in a contiguous block of memory. Each element in the array is accessed by its index, which is an integer value that represents its position within the array.

For example, consider the following array:

```
int array[5] = {1, 2, 3, 4, 5};
```

This array has five elements, which are stored in memory as shown below:

```
Index: 0 1 2 3 4
Value: 1 2 3 4 5
```

To access an element in the array, we use its index in square brackets. For example, to access the third element (which has a value of 3), we would use the following code:

```
int thirdElement = array[2]; // thirdElement is now 3
```

We can also modify the elements in the array using the same notation:

```
array[3] = 10; // The fourth element of the array is now 10
```

It is important to note that arrays in C++ are zero-indexed, meaning that the first element has an index of 0, the second element has an index of 1, and so on.

Pointer: A data type that stores the memory address of another variable. It is a variable that stores the memory address of another variable. It allows you to manipulate the value of a variable indirectly by accessing the memory location of the variable.
To declare a pointer in C++, you use the * operator followed by the variable name. For example:

```
int *ptr;
```

This declares a pointer called "ptr" that points to an integer variable.
To assign the address of a variable to a pointer, you use the & operator. For example:

```
int x = 10;
int *ptr = &x;
```

This assigns the memory address of the variable "x" to the pointer "ptr".
To access the value of a variable through a pointer, you use the * operator. For example:

```
int y = *ptr;
```

This assigns the value of the variable pointed to by "ptr" to the variable "y".
Pointers can be used to dynamically allocate memory for variables in C++ using the new operator. This allows you to create variables that are not fixed in size and can be modified during runtime.

For example:

```
int *ptr = new int;
*ptr = 10;
```

This dynamically allocates memory for an integer variable and assigns the value of 10 to it through the pointer "ptr".
Pointers can be useful in C++ for optimizing code performance and for working with large data sets. However, they can also be dangerous if not used correctly, as they can lead to memory leaks and segmentation faults. It is important to carefully consider when to use pointers in your C++ code.
Structure: a data type that allows you to define a composite data type, consisting of several different data types. A structure in C++ is a user-defined data type that allows you to combine different data types into a single unit. It is similar to a class, but the main difference is that a structure does not have any member functions.

Here is an example of how to define a structure in C++:

```
struct Point {
  int x;
```

```
  int y;
};
```

This creates a structure called "Point" with two members: "x" and "y". Both members are of type "int". You can access the members of a structure using the "dot" operator. For example:

```
Point p;
p.x = 10;
p.y = 20;
```

You can also create an instance of a structure using the "new" operator:

```
Point *p = new Point;
p->x = 10;
p->y = 20;
```

You can also define functions that operate on structures. For example:

```
void printPoint(Point p) {
  cout << "(" << p.x << ", " << p.y << ")" << endl;
}
```

Structures can also have constructors and destructors, similar to classes.

Overall, structures are useful for organizing data in a way that is easy to access and manipulate. They are often used in combination with classes to create more complex data structures.

Enum

Enum in C++ is a data type that allows you to define a set of named integer constants. It is often used to represent a set of related constants that represent a particular value or state. Enum is often used to improve the readability and maintainability of code by giving names to constants that would otherwise be represented by magic numbers. It also helps to prevent errors by catching typos and other mistakes at compile time.

For example, consider the following enum:

```
enum Days {
Sunday,
Monday,
Tuesday,
Wednesday,
Thursday,
Friday,
Saturday
};
```

This enum defines a set of named constants that represent the days of the week. Each constant is given a unique integer value, starting with 0 for Sunday and ending with 6 for Saturday.

You can then use the constants in your code like this:

```
Days today = Monday;
if (today == Wednesday) {
// do something
}
```

Enum can also be given explicit values, like this:

```
enum Days {
Sunday = 1,
Monday = 2,
Tuesday = 3,
Wednesday = 4,
Thursday = 5,
Friday = 6,
Saturday = 7
};
```

In this case, the constant Sunday will have a value of 1, Monday will have a value of 2, and so on.

OPERATORS AND EXPRESSIONS IN C++

Operators in C++ are symbols that perform specific actions on one or more operands (variables or values). In C++, there are several types of operators that you can use to perform operations on variables and expressions:

Arithmetic operators

These operators are used to perform mathematical calculations like addition (+), subtraction (-), multiplication (*), division (/), and modulus (%).

```
int x = 10 + 10;          // return 20
int x = 10 * 10;          // return 100
int x = 10 / 10;          // return 1
int x = 10 % 10;          // return 0
```

Comparison operators

These operators are used to compare two values and return a boolean value (true or false). Examples include equal to (==), not equal to (!=), greater than (>), and less than (<).

```
int x = 5;
int y = 3;
cout << (x > y); // returns 1 (true) because 5 is greater than 3
```

Logical operators

These operators are used to perform logical operations like AND (&&), OR (||), and NOT (!).

Assignment operators

These operators are used to assign a value to a variable. Examples include the equals sign (=) and the compound assignment operator (+=, -=, *=, /=, %=, <<=, >>=, &=, ^=, |=)

Operator	**Example**	**Same As**
=	x = 10	x = 10
+=	x += 5	x = x + 5

+-	x -= 3	x = x -3
*=	x *= 3	x = x * 3
/=	x /= 2	x = x / 2
%=	x %= 2	x = x % 2
&=	x &= 3	x = x & 3
|=	x *= 3	x = x * 3
^=	x *= 3	x = x * 3
>>=	x <<= 3	x = x << 3
<<=	x<<=3	x=x<<3

Bitwise operators

They are used to perform bit-level operations on variables. They operate on the individual bits of a variable, rather than on the variable as a whole. Here is a list of the most commonly used bitwise operators in C++:

- & (bitwise AND)
- | (bitwise OR)
- ^ (bitwise XOR)
- ~ (bitwise NOT)
- << (left shift)
- >> (right shift)

Here is an example that demonstrates the use of bitwise operators in C++:

```
#include <iostream>

int main()
{
  int x = 10; // 1010 in binary
  int y = 20; // 10100 in binary

  std::cout << "x & y = " << (x & y) << std::endl; // 0010 = 0
  std::cout << "x | y = " << (x | y) << std::endl; // 11110 = 30
```

```
 std::cout << "x ^ y = " << (x ^ y) << std::endl; // 11000 = 30
 std::cout << "~x = " << ~x << std::endl; // 11111111111111111111111111110101 = -11
 std::cout << "x << 2 = " << (x << 2) << std::endl; // 101000 = 40
 std::cout << "x >> 1 = " << (x >> 1) << std::endl; // 101 = 5

 return 0;
}
```

Output:

```
x & y = 0
x | y = 30
x ^ y = 30
~x = -11
x << 2 = 40
x >> 1 = 5
```

Miscellaneous operators

These are various other operators that don't fit into any of the above categories. For example: sizeof, ?:, ->, .*, ::, ..

Increment and decrement operators

These operators are used to increase or decrease the value of a variable by one. The increment operator is ++ and the decrement operator is --.

```
#include <iostream>
int main() {
 int x = 5;
 int y = 10;

 // Arithmetic operators
 int sum = x + y;
 int difference = x - y;
 int product = x * y;
 int quotient = y / x;
 int remainder = y % x;
```

```
    std::cout << "x + y = " << sum << std::endl;
    std::cout << "x - y = " << difference << std::endl;
    std::cout << "x * y = " << product << std::endl;
    std::cout << "y / x = " << quotient << std::endl;
    std::cout << "y % x = " << remainder << std::endl;

    // Comparison operators
    bool isEqual = (x == y);
    bool isNotEqual = (x != y);
    bool isGreater = (x > y);
    bool isLess = (x < y);
    bool isGreaterOrEqual = (x >= y);
    bool isLessOrEqual = (x <= y);

    std::cout << "x == y is " << isEqual << std::endl;
    std::cout << "x != y is " << isNotEqual << std::endl;
    std::cout << "x > y is " << isGreater << std::endl;
    std::cout << "x < y is " << isLess << std::endl;
    std::cout << "x >= y is " << isGreaterOrEqual << std::endl;
    std::cout << "x <= y is " << isLessOrEqual << std::endl;

    // Logical operators
    bool isTrue = true;}
    bool isFalse = false;
    bool result1 = isTrue && isFalse;
    bool result2 = isTrue || isFalse;
    bool result3 = !isFalse;

    std::cout << "true && false is " << result1 << std::endl;
    std::cout << "true || false is " << result2 << std::endl;
    std::cout << "!false is " << result3 << std::endl;

    return 0;
}
```

This program will output the following:

```
x + y = 15
```

Here is an example that demonstrates the use of some of these operators in C++:

```
#include <iostream>

int main()
{
  int x = 10;
  int y = 20;
  int z = 30;

  // Arithmetic operators
  std::cout << "x + y = " << x + y << std::endl;
  std::cout << "x - y = " << x - y << std::endl;
  std::cout << "x * y = " << x * y << std::endl;
  std::cout << "y / x = " << y / x << std::endl;
  std::cout << "y % x = " << y % x << std::endl;

  // Relational operators
  std::cout << "x > y = " << (x > y) << std::endl;
  std::cout << "x < y = " << (x < y) << std::endl;
  std::cout << "x >= y = " << (x >= y) << std::endl;
  std::cout << "x <= y = " << (x <= y) << std::endl;
  std::cout << "x == y = " << (x == y) << std::endl;
  std::cout << "x != y = " << (x != y) << std::endl;

  // Logical operators
  std::cout << "x > y && x < z = " << ((x > y) && (x < z)) << std::endl;
  std::cout << "x > y || x < z = " << ((x > y) || (x < z)) << std::endl;
  std::cout << "!(x > y) = " << !(x > y) << std::endl;
```

ASSIGNMENT 1:

1. Write a C++ program that takes in two integer values from the user and calculates the sum, difference, product, and quotient using the appropriate operators.
2. Write a C++ program that calculates the area of a circle given the radius as input from the user. Use the formula pi*r^2 and use the exponentiation operator to calculate the squared radius.
3. Write a C++ program that takes in three integer values from the user and checks if they are equal using the equality operator. If they are equal, print "Equal" to the screen, otherwise print "Not equal".
4. Write a C++ program that reads in a string from the user and checks if it is equal to the string "hello" using the equality operator. If it is equal, print "Hello!" to the screen, otherwise print "Goodbye!".
5. Write a C++ program that reads in two integer values from the user and checks if the first value is greater than the second using the greater than operator. If it is, print "First value is greater" to the screen, otherwise print "Second value is greater".
6. Write a C++ program that reads in a character from the user and checks if it is an uppercase letter using the logical AND operator and the isupper() function. If it is, print "Uppercase letter" to the screen, otherwise print "Not an uppercase letter".
7. Write a C++ program that reads in a string from the user and checks if it contains the character 'a' using the in operator. If it does, print "Contains 'a'" to the screen, otherwise print "Does not contain 'a'".
8. Write a C++ program that reads in an integer value from the user and checks if it is odd using the modulus operator. If it is odd, print "Odd number" to the screen, otherwise print "Even number".
9. Explain the difference between the assignment operator and the equality operator in C++. Give an example of each.
10. Write a C++ program that uses the ternary operator to determine the larger of two integers.
11. Explain the difference between the increment and decrement operators in C++. Give an example of each.
12. Write a C++ program that uses the logical AND operator to check if a number is both odd and divisible by 3.
13. Write a C++ program that uses the bitwise XOR operator to swap the values of two variables.
14. Explain the difference between the left shift and right shift operators in C++. Give an example of each.
15. Write a C++ program that uses the conditional operator to determine the larger of two floating point numbers.
16. Explain the difference between the bitwise NOT and logical NOT operators in C++. Give an example of each.

17. Write a C++ program that uses the bitwise AND operator to check if a number is a power of 2.
18. Write a C++ programming to convert binary to decimal.
19. Write a C++ programming calculator to convert binary to decimal, binary to hexadecimal, binary to octadecimal.

CONTROL STRUCTURES

Control structures in C++ programming are statements that allow a programmer to control the flow of execution of their code. For example, an if statement allows a program to make a decision based on a certain condition. These include:

1. **Conditional statements:** If-else and switch statements allow a programmer to execute different blocks of code depending on whether a certain condition is met or not.
2. **Jump statements:** Break, continue, and goto statements allow a programmer to jump to a different point in their code, either to terminate a loop or skip over certain blocks of code.

Control structures are an important part of any programming language, as they allow a programmer to write code that can adapt to changing circumstances and make decisions based on input or other conditions. There are several control structures in C++ that allow the programmer to control the flow of the program.
These control structures include:

If Statements

If statements in C++ programming allow the program to execute a certain block of code only if a certain condition is met.

For example 1:

```
if (x > y)
{
cout << "x is greater than y" << endl;
}
```

This code will only execute the cout statement if x is indeed greater than y. If statements can also include an else clause to execute a different block of code if the condition is not met: If statements can also include multiple conditions using the "else if" clause:

```
if (x > y)
{
    cout << "x is greater than y" << endl;
}
else if (x == y)
```

```
{
    cout << "x is equal to y" << endl;
}
else
{
    cout << "x is less than y" << endl;
}
```

In this case, if x is greater than y, the first block of code will execute. If x is not greater than y but is equal to y, the second block of code will execute. If x is neither greater than y nor equal to y, the third block of code will execute.

For example 2:

```
if (x > 5)
{
// code to execute if x is greater than 5
}
```

if-else: This control structure allows the programmer to execute a block of code based on a condition. If the condition is true, the block of code within the if statement is executed, otherwise the block of code within the else statement is executed.

```
#include <iostream>
using namespace std;

int main() {
int x = 5;
int y = 10;

if (x > y)
{
    cout << "x is greater than y." << endl;
} else
{
    cout << "x is not greater than y." << endl;
}
```

```
return 0;
}
```

// Output: x is not greater than y.

Switch-case:

A switch case is a control statement in C++ that allows a programmer to execute different blocks of code based on the value of a variable or expression. It is similar to an if-else statement, but it is more efficient for evaluating multiple cases. Switch case statements are useful when there are multiple possible values for a variable and you want to perform different actions based on those values. They can be used to simplify complex if-else statements and make the code more readable.

The programmer can specify multiple cases for different values of the variable and a default case for any other value. They are often used to implement multi-way branching, where the program needs to take different actions based on the value of a particular variable.

The syntax for switch case in C++ is as follows:

Example 1

```
switch (variable) {
case value1:
// code to execute if variable == value1
break;
case value2:
// code to execute if variable == value2
break;
...
default:
// code to execute if variable does not match any of the values
}
```

In the above syntax, the variable is the expression being evaluated, and the case statements are the different values that the variable can take. When the variable is evaluated, the code in the

corresponding case block is executed. If the variable does not match any of the values, the code in the default block is executed.

Example 2:

```
switch (expression) {
case value1:
// code to be executed if expression == value1
break;
case value2:
// code to be executed if expression == value2
break;
// ...
default:
// code to be executed if expression does not match any of the above cases
break;
}
```

The expression is evaluated and compared to each of the case values. If a match is found, the corresponding code block is executed. If no match is found, the code in the default block is executed. It is important to include the break statement at the end of each case block, as it exits the switch-case statement and prevents the code in the following cases from being executed.

For example, consider the following switch-case statement:

```
int num = 3;

switch (num) {
case 1:
    std::cout << "Number is 1" << std::endl;
break;
case 2:
    std::cout << "Number is 2" << std::endl;
break;
case 3:
    std::cout << "Number is 3" << std::endl;
break;
default:
    std::cout << "Number is not 1, 2, or 3" << std::endl;
break;
```

```
}
```

In this example, the value of num is 3, so the code in the **case 3 block** is executed, resulting in the output "Number is 3".

FOR LOOP

The for loop in C++ programming is a control flow statement that allows you to repeat a block of code a certain number of times. It consists of a loop variable, a loop condition, and a loop increment or decrement. The loop variable is initialized to a starting value, and then the loop condition is checked. If the condition is true, the code block inside the loop is executed.

After the code block is executed, the loop variable is updated according to the increment or decrement specified in the loop. The loop condition is checked again, and if it is still true, the code block is executed again. This process continues until the loop condition is false, at which point the loop ends and control is transferred to the next line of code outside of the loop. The basic syntax of a for loop in C++ is:

```
for (initialization; condition; update) {
// code block to be executed
}
```

The programmer specifies the number of iterations and the loop variables.

```
for (int i = 0; i < 10; i++) {
cout << i << endl;
}
```

// This loop will print the numbers 0 through 9 on separate lines. The variable i is initialized to 0, and the loop continues until i is no longer less than 10. The variable i is incremented by 1 each time the loop iterates.

```
#include <iostream>

 using namespace std;

int main()
{
 // Declaring variables
 int num, sum = 0;

 // Inputting number of iterations
 cout << "Enter the number of iterations: ";
 cin >> num;

 // Using for loop to iterate and add numbers
 for (int i = 1; i <= num; i++)
 {
    sum += i;
 }
 cout << "The sum of the numbers from 1 to " << num << " is: " << sum << endl;

 return 0;
}
```

//Displaying the result

```
The sum of the numbers from 1 to 5 is: 15
```

Example 2

```
#include <iostream>
using namespace std;

int main() {
 for (int i = 0; i < 5; i++) {
    cout << i << "\n";
 }
 return 0;
}
```

While Loop

A while loop in C++ is a type of loop that executes a block of code repeatedly as long as a certain condition is met/true.
The syntax for a while loop in C++ is:

```
while (condition) {
// code to be executed
}
```

The condition is evaluated before the loop is entered. If the condition is true, the code inside the loop is executed. If the condition is false, the loop is skipped and control is transferred to the next statement after the loop. The loop will continue to execute until the condition becomes false.

This while loop will print out the numbers 0 through 9, incrementing the value of i by 1 each time the loop is executed. The loop will continue to execute as long as i is less than 10.

Here is an example of a while loop in C++:

```
int i = 0;
while (i < 10) {
cout << i << " ";
i++;
}
```

This will print out

```
0 1 2 3 4 5 6 7 8 9
```

Here is another example of a while loop in C++ that counts down from 10 to 1:

```
#include <iostream>
using namespace std;

int main() {
int count = 10;
while (count > 0) {
cout << count << endl;
count--;
}
return 0;
}
```

This will print out

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

This while loop will print the numbers 10 through 1 on separate lines. The condition (i > 0) is checked at the beginning of each iteration, and as long as it is true, the loop will continue. Once i reaches 0, the condition is false and the loop will exit.

Do-while Loop

do-while loop: This control structure is similar to the while loop, but the block of code is executed at least once before the condition is checked. A do-while loop in C++ is a type of loop that will execute a block of code at least once, and then continue to repeat the block of code as long as a certain condition is true. The syntax for a do-while loop in C++ is as follows:

```
do {
// code to be executed
} while (condition);
```

The block of code within the do { } block will be executed first, and then the condition within the while ( ) clause will be evaluated. If the condition is true, the block of code will be executed again. If the condition is false, the loop will terminate.

Here is an example of a do-while loop in C++:

```
int x = 0;
do  {
    cout << "x is: " << x << endl;
        x++;
}
while (x < 10);
```

x is: 0 x is: 1 x is: 2 x is: 3 x is: 4 x is: 5	This loop will output the value of x to the console and then increment x by 1 each time it iterates. The loop will continue to execute until x is equal to 10, at which point the

x is: 6 x is: 7 x is: 8 x is: 9	condition (x < 10) will be false and the loop will terminate.

```
#include <iostream>

int main()
{
    int counter = 0; // initialize the counter variable

    do
    {
   std::cout << "Counter: " << counter << std::endl;
   counter++; // increment the counter
    }
    while (counter < 10);
    // continue looping as long as counter is less than 10

return 0;
}
```

/* Output:

```
Counter: 0
Counter: 1
Counter: 2
Counter: 3
Counter: 4
Counter: 5
Counter: 6
Counter: 7
Counter: 8
Counter: 9
```

*/

Break Statement

The break statement is a control flow statement in C++ that is used to terminate the execution of a loop or switch statement prematurely. It is typically used to exit a loop when a certain condition is met or to switch to a different case in a switch statement. When the break statement is encountered, the program will immediately exit the loop or switch statement and continue execution at the next line of code.

When the break statement is encountered, it immediately exits the loop or switch statement and continues executing the code that follows. This can be useful for breaking out of a loop or switch statement early when certain conditions are met, or for breaking out of a loop when an error or other exception occurs.

```
#include <iostream>

int main()
{
int counter = 0; // initialize the counter variable

Copy code
do
{
    std::cout << "Counter: " << counter << std::endl;
    counter++; // increment the counter
} while (counter < 10); // continue looping as long as counter is less than 10

return 0;
}
```

/* Output:

```
Counter: 0
Counter: 1
Counter: 2
Counter: 3
Counter: 4
Counter: 5
Counter: 6
Counter: 7
Counter: 8
Counter: 9
```

*/

Here are a few examples of using the "break" statement in C++:

Breaking out of a loop:

```
for (int i = 0; i < 10; i++)
{
    if (i == 5)
    {
        break; // exit the loop when i is equal to 5
    }
    cout << i << " ";
}
// Output: 0 1 2 3 4
```

Breaking out of a switch statement:

```
int num = 2;

switch (num) {
case 1:
    cout << "One" << endl;
break;
case 2:
    cout << "Two" << endl;
break;
case 3:
    cout << "Three" << endl;
break;
default:
cout << "Invalid number" << endl;
break;
}
```

// Output: Two

Breaking out of nested loops:

```
for (int i = 0; i < 10; i++)
{
    for (int j = 0; j < 10; j++)
    {
        if (i == 5 && j == 5)
        {
            break; // exit both loops when i and j are both equal to 5
        }
    cout << i << " " << j << " ";
    }
}
```

// Output: 0 0 1 0 2 0 3 0 4 0

Breaking out of a labeled block:
label:

```
for (int i = 0; i < 10; i++)
{
    for (int j = 0; j < 10; j++)
    {
        if (i == 5 && j == 5) {
            break label;
            // exit both loops when i and j are both equal to 5
        }
    cout << i << " " << j << " ";
    }
}
```

// Output: 0 0 1 0 2 0 3 0 4 0

CLASSWORK

1. How does the break statement work in a for loop in C++?
2. What happens if you use the break statement within a nested loop in C++?
3. How can you exit a switch statement in C++ using the break statement?
4. Can you use the break statement to exit a do-while loop in C++?
5. How do you use the break statement to exit a loop within a function in C++?
6. Can you use the break statement to exit multiple loops at once in C++?
7. How do you use the break statement to exit a loop and skip certain code in C++?
8. Can you use the break statement to exit a loop and return a value in a function in C++?
9. How do you use the break statement to exit a loop and jump to a specific point in your code in C++?
10. Can you use the break statement in a infinite loop in C++ to exit the loop?
11. What is the purpose of the break statement in C++?
12. How does the break statement allow a program to exit a loop?
13. Can the break statement be used to exit a switch statement in C++?
14. In what cases would you use the break statement in a for loop?
15. How can you use the break statement to exit a nested loop in C++?
16. Can you use a break statement to exit a function in C++?
17. How do you label a loop in C++ to allow the break statement to exit it?
18. Can you use a break statement to exit multiple loops at once in C++?
19. How can you prevent a break statement from exiting a loop in C++?
20. How does the continue statement differ from the break statement in C++?

ASSIGNMENT 2

1. Write a c++ program that reads in a series of integers from the user until a negative number is entered. Once a negative number is entered, the program should print the sum of all the positive numbers entered.

```
#include <iostream>

using namespace std;

int main() {
int sum = 0; // Initialize the sum to 0
int num; // Declare the input variable
```

```
// Read in integers from the user until a negative number is entered
cout << "Enter a series of integers (enter a negative number to stop): ";
while (true) {
cin >> num; // Read in the integer
if (num < 0) break; // If it's negative, stop the loop
sum += num; // Otherwise, add it to the sum
}

// Print the sum of all the positive numbers
cout << "The sum of all the positive numbers is: " << sum << endl;

return 0;
}
```

2. Write a c++ program that reads in a series of strings from the user until the string "done" is entered. The program should then print out the number of strings entered.

```
#include <iostream>
#include <string>

int main() {
  std::string input;
  int count = 0;

  // Read in a series of strings from the user
  while (true) {
    std::cout << "Enter a string (enter 'done' to stop): ";
    std::cin >> input;

    // Break out of the loop if the user enters "done"
    if (input == "done") {
      break;
    }

    // Increment the count for each string entered
    count++;
```

```
  }

  // Print out the number of strings entered
  std::cout << "Number of strings entered: " << count << std::endl;

  return 0;
}
```

This program reads in a string from the user using the cin object, which is part of the iostream library. It then checks if the input string is equal to "done" using the == operator. If it is, the loop is broken using the break statement. Otherwise, the count is incremented using the ++ operator. Finally, the program prints out the number of strings entered using the cout object.

3. Write a c++ program that reads in a series of numbers from the user until the number 0 is entered. The program should then print out the largest number entered.

```
#include <iostream>
using namespace std;

int main() {
// Declare variables
int num, largest;

// Read in first number
cout << "Enter a number (0 to exit): ";
cin >> num;

// Set largest to first number
largest = num;

// Read in remaining numbers
while (num != 0) {
    cout << "Enter a number (0 to exit): ";
    cin >> num;
    if (num > largest) { // If current number is larger than largest
        largest = num; // Update largest
    }
}

// Print out largest number
cout << "Largest number: " << largest << endl;
```

```
return 0;
}
```

4. Write a c++ program that reads in a series of characters from the user until the character 'q' is entered. The program should then print out the number of vowels entered.

```
#include <iostream>
using namespace std;

int main() {
char ch;
int vowel_count = 0; // initialize vowel count to 0

cout << "Enter a series of characters, enter 'q' to stop: ";
cin >> ch; // read in first character

while (ch != 'q') { // loop until 'q' is entered
if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u' || ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U') {
vowel_count++; // increment vowel count if character is a vowel
}
cin >> ch; // read in next character
}

cout << "Number of vowels entered: " << vowel_count << endl; // print out vowel count

return 0;
}
```

Alternatively:

```
#include <iostream>

int main() {
char c; // variable to store user input
int vowel_count = 0; // variable to store vowel count

std::cout << "Enter a series of characters (enter 'q' to quit): ";
```

```
std::cin >> c; // read in first character

while (c != 'q') { // loop until 'q' is entered
if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u') { // check if character is a vowel
vowel_count++; // increment vowel count
}
std::cin >> c; // read in next character
}

std::cout << "Number of vowels entered: " << vowel_count << std::endl; // print vowel count

return 0;
}
```

5. Write a C++ program that reads in a series of numbers from the user until a number that is not a multiple of 3 is entered. The program should then print out the sum of all the multiples of 3 entered.

```
#include <iostream>
using namespace std;

int main() {
int num;
int sum = 0;

cout << "Enter a series of numbers, enter a number that is not a multiple of 3 to stop:" << endl;
cin >> num;

// keep reading in numbers until one that is not a multiple of 3 is entered
    while (num % 3 == 0) {
        sum += num; // add the multiple of 3 to the sum
        cin >> num;
    }

cout << "The sum of all the multiples of 3 entered is: " << sum << endl;

return 0;
}
```

6. How do you use the switch statement in C++? Give an example.
7. What is the purpose of the break statement in a loop? Give an example of its usage.
8. How does the continue statement work in a loop? Give an example of its usage.
9. Explain the difference between the do-while and while loops in C++. When would you use each one?
10. What is the output of the following code:

```
int i = 1;
while (i <= 10)
{
    if (i % 2 == 0)
    {
        cout << i << " ";
    }
  i++;
}
```

11. Write a for loop that counts down from 10 to 1, printing each number.
12. Write a program that asks the user to input a number, and then uses a loop to print out the numbers from 1 to that number. The program should only print out odd numbers.
13. Write a program that uses a loop to calculate the sum of the first 10 positive integers.
14. Write a program that uses a loop to find the smallest number in an array of integers.
15. What is the output of the following code:

```
int x = 5;
if (x < 10)
{
    cout << "x is less than 10" << endl;
}
    else if (x == 10)
{
    cout << "x is equal to 10" << endl;
}
    else
{
    cout << "x is greater than 10" << endl;
}
```

FUNCTION IN C++ PROGRAMMING

Learning outcomes:
The learner will be able to

- understand the concept of functions
- use functions in C++.
- write code to declare a function,
- understand return type function,
- use function arguments.

Introduction

Functions are a fundamental concept in programming that helps to improve the readability, maintainability, and efficiency of the code. They are blocks of code that perform a specific task and can be reused throughout a program. Functions can take input in the form of arguments and can return output in the form of a return value.

Functions are used to organize and structure code, making it more modular, readable and easy to maintain. They allow you to reuse the same code multiple times throughout your program, making it more efficient and less error-prone. The concept of function is to divide a complex problem into small and manageable subproblems. Each subproblem is then solved by a function. This way, it becomes easy to understand and debug the code. Functions also make it possible to write code that is more reusable, thus reducing the amount of redundant code.

Function Declaration in C++

To declare a function in C++, the keyword "void" is used if the function does not return a value, otherwise, a specific data type such as "int" or "string" is used. The function's name, along with its parameters, follows the keyword.

Function in C++ must have return type, the name of the function, and the parameters it takes. The general syntax for declaring a function is:

For example, the following code declares a function named "greet" that takes in a string argument and does not return a value:

```
void greet(string name) {
    cout << "Hello, " << name << endl;
}
```

- **return_type** is the data type of the value that the function returns. If the function does not return a value, the keyword void is used.
- **function_name** is the name of the function, which should be descriptive and indicative of the task the function performs.
- **parameter_list** is a list of variables that the function takes as input.

DIFFERENT TYPES OF FUNCTIONS AND HOW TO DECLARE THEM IN C++:

1. A function that takes no arguments and returns no value:

```
void printGreeting() {
    cout << "Hello, World!" << endl;
}
```

2. A function that takes an integer argument and returns an integer value:

```
int square(int x) {
    return x * x;
}
```

3. A function that takes two string arguments and returns a string value:

```
string concatenate(string a, string b) {
    return a + b;
}
```

4. A function that takes an array of integers and its size as arguments and returns the sum of its elements:

```
int sumArray(int arr[], int size) {
    int sum = 0;
    for (int i = 0; i < size; i++) {
        sum += arr[i];
    }
    return sum;
}
```

It's worth noting that you can also use reference and pointer parameters in the functions. This allows you to change the values of the original variables passed to the function.

In summary, to declare a function in C++, you need to specify the return type, the name of the function, and the parameters it takes. You should give the function a name that is descriptive and indicative of the task it performs, and use the appropriate data types for the return value and parameters.

Return Types Function

A return type in a function is the data type of the value that the function returns. In C++, a function can return any data type, such as int, double, string, or even a custom struct or class. The return type is specified before the function name in the function declaration. If a function does not return a value, the keyword void is used as the return type.

For example, the following code declares a function named "greet" that takes in a string argument and does not return a value:

```
void greet(string name) {
    cout << "Hello, " << name << endl;
}
```

And the following code declares a function named "add" that takes in two integer arguments and returns the sum of them.

```
int add(int a, int b) {
    return a + b;
}
```

The return statement is used to return a value from a function. It can be used with any data type. It must be the last statement in the function body, as the program execution jumps to the point from where the function was called after the return statement.

```
int add(int a, int b) {
    int c = a+b;
    // other statements
    return c;
}
```

It's worth noting that if a function is declared as "void" it doesn't have to return anything and the return statement is optional in this case.
In summary, a return type in a function is the data type of the value that the function returns. It is specified before the function name in the function declaration. The return statement is used to return a value from a function and it must be the last statement in the function body. If a function does not return a value, the keyword void is used as the return type.

Use Function Arguments

Function arguments, also known as parameters, are the values that are passed to a function when it is called. These values are used as input for the function, and are specified within the parentheses of the function declaration.
For example, the following code declares a function named "greet" that takes in a string argument:

```
void greet(string name) {
    cout << "Hello, " << name << endl;
}
```

To call this function, you would pass a string value as an argument, like this:

```
greet("John");
```

Function arguments can be of any data type, such as int, double, string, or even a custom struct or class. Functions can also take multiple arguments, and these arguments can be of different data types. It is important to note that the order of the arguments in the function declaration must match the order of the arguments when the function is called.

You can also pass variables as arguments to a function. For example, the following code declares a function named "add" that takes in two integer arguments and returns the sum of them.

```
int add(int a, int b) {
    return a + b;
}
```

Then you can call the function like this:

```
int x = 1, y = 2;
int result = add(x, y);
cout << result; // Prints 3
```

In C++, function arguments can be passed by value, by reference or by pointer.

- When passed by value, a copy of the argument is passed to the function and any changes made to the argument inside the function have no effect on the original variable.
- When passed by reference, a reference to the original variable is passed to the function, so any changes made to the argument inside the function are reflected in the original variable.
- When passed by a pointer, a pointer to the original variable is passed to the function, so any changes made to the argument inside the function are reflected in the original variable.

Note that when you pass a large object or array as an argument to a function, it's more efficient to pass it by reference or pointer to avoid unnecessary copying.
In summary, function arguments, also known as parameters, are the values that are passed to a function when it is called. These values are used as input for the function, and are specified within the parentheses of the function declaration. Functions can take multiple arguments, and these arguments can be of different data types. They can be passed by value, by reference or by pointer.

CLASSWORK (USING FUNCTION)

These exercises are designed to help you practice different aspects of working with functions in C++, such as declaring functions, using function arguments, and using return values. You are encouraged to test the function with different inputs and debug their code if necessary.

1. Write a function that takes an integer as an argument and returns the absolute value of the number.

```
int absolute(int num) {
    if (num < 0) {
        return -num;
    }
    return num;
}
```

2. Write a function that takes two integers as arguments and returns the maximum of the two numbers.

```
int max(int x, int y) {
    if (x > y) {
        return x;
    }
    return y;
}
```

3. Write a function that takes an array of integers and its size as arguments, and returns the average of the array elements.

```
double average(int arr[], int size) {
    double sum = 0;
    for (int i = 0; i < size; i++) {
        sum += arr[i];
    }
    return sum / size;
}
```

4. Write a function that takes a character as an argument and returns whether it is a vowel or not.

```
bool isVowel(char c) {
    c = tolower(c);
    return (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u');
}
```

5. Write a function that takes a string as an argument and returns the number of vowels in the string.

```
int countVowels(string str) {
    int count = 0;
    for (int i = 0; i < str.length(); i++) {
        if (isVowel(str[i])) {
            count++;
        }
    }
    return count;
}
```

These practice activities are designed to help electrical engineering students apply the concepts of functions in C++ to real-world electrical engineering problems

6. Write a function that takes a voltage and a resistance as arguments, and returns the current flowing through the circuit. The current can be calculated using Ohm's Law, which states that I = V / R.

```
double current(double voltage, double resistance) {
    return voltage / resistance;
}
```

7. Write a function that takes an input voltage and a transfer function as arguments, and returns the output voltage using the formula: Vout = Vin * tf.

```
double transferFunction(double Vin, double tf) {
    return Vin * tf;
}
```

8. Write a function that takes a power and a voltage as arguments, and returns the apparent power in VA using the formula: VA = P * V.

```
double apparentPower(double power, double voltage) {
    return power * voltage;
}
```

9. Write a function that takes a frequency and an inductance as arguments, and returns the inductive reactance in ohms using the formula: XL = 2π * f * L.

```
double inductiveReactance(double frequency, double inductance) {
    return 2 * M_PI * frequency * inductance;
}
```

10. Write a function that takes a frequency and a capacitance as arguments, and returns the capacitive reactance in ohms using the formula: XC = 1 / (2π * f * C).

```
double capacitiveReactance(double frequency, double capacitance) {
    return 1 / (2 * M_PI * frequency * capacitance);
}
```

Activities:

Write a function to calculate the area of a triangle in c++

```
#include <iostream>
#include <cmath>

using namespace std;

double calculateTriangleArea(double base, double height)
{
return 0.5 * base * height;
}

int main()
```

```
{
    double base, height;
    cout << "Enter base of triangle: ";
    cin >> base;
    cout << "Enter height of triangle: ";
    cin >> height;

    cout << "Area of triangle: " << calculateTriangleArea(base, height) <<                endl;
return 0;
}
```

Write a function in c++ to convert temperature from degrees to Fahrenheit

```
#include <iostream>
using namespace std;

double convertToFahrenheit(double degrees) {
double fahrenheit = (degrees * 9/5) + 32;
return fahrenheit;
}

int main() {
    double degrees = 0;
    cout << "Enter temperature in degrees: ";
    cin >> degrees;
    double fahrenheit = convertToFahrenheit(degrees);
    cout << degrees << " degrees is equal to " << fahrenheit << "          Fahrenheit." << endl;
return 0;
}
```

Write a function in c++ to calculate the resistance of a circuit

```
#include <iostream>

double calcResistance(double voltage, double current) {
return voltage / current;
}

int main() {
```

```
double voltage, current;
std::cout << "Enter voltage: ";
std::cin >> voltage;
std::cout << "Enter current: ";
std::cin >> current;
std::cout << "Resistance: " << calcResistance(voltage, current) << std::endl;
return 0;
}
```

CLASSWORK

1. How do you define a function in C++?
2. What is a function prototype and why is it necessary?
3. How do you pass arguments to a function in C++?
4. How do you return a value from a function in C++?
5. Can a function have multiple return statements?
6. What is the difference between a void function and a non-void function in C++?
7. How do you define default values for function parameters in C++?
8. Can a function call itself recursively in C++? If so, how do you do it?
9. What is the difference between a function call and a function definition in C++?
10. How do you create and use a function pointer in C++?

ASSIGNMENT 3:

1. Write a function in C++ that takes in two integers, x and y, and returns the sum of x and y.
2. Write a function in C++ that takes in a string and returns the number of vowels in the string.
3. Write a function in C++ that takes in a list of integers and returns the average of all the integers in the list.
4. Write a function in C++ that takes in a sentence and returns the sentence with all the vowels removed.
5. Write a function in C++ that takes in a list of strings and returns the longest string in the list.
6. Write a function in C++ that takes in a list of integers and returns the median value.
7. Write a function in C++ that takes in a list of integers and returns a new list with all the even numbers removed.
8. Write a function in C++ that takes in a list of strings and returns a new list with all the strings sorted in alphabetical order.
9. Write a function in C++ that takes in a list of integers and returns the sum of all the integers in the list.
10. Write a function in C++ that takes in a list of strings and returns a new list with all the strings reversed.

ASSIGNMENT 4

PRACTICAL C++ QUESTIONS FOR ELECTRICAL ENGINEERING STUDENTS

1. Write a C++ program to simulate the behavior of a simple electric circuit. The program should prompt the user for the resistance, inductance, and capacitance values of the circuit, and then compute and print out the impedance and phase angle of the circuit at a given frequency.
2. Write a C++ program to calculate the power consumed by a resistor in an electric circuit. The program should prompt the user for the voltage, current, and resistance values of the circuit, and then compute and print out the power consumed by the resistor.
3. Write a C++ program to solve a system of linear equations using Gaussian elimination. The program should prompt the user for the number of equations and variables in the system, and then allow the user to input the coefficients and constants of the equations. The program should then solve the system and print out the solutions.
4. Write a C++ program to calculate the electric field and potential at a point in space due to a charged particle. The program should prompt the user for the charge, position, and velocity of the particle, and then compute and print out the electric field and potential at a given point in space.
5. Write a C++ program to simulate the behavior of a simple alternating current (AC) circuit. The program should prompt the user for the resistance, inductance, and capacitance values of the circuit, and then compute and print out the impedance, phase angle, and power factor of the circuit at a given frequency.
6. Write a C++ program to simulate the behavior of a simple DC motor. The program should prompt the user for the resistance, inductance, and torque constants of the motor, and then compute and print out the current, speed, and power of the motor for a given applied voltage.
7. Write a C++ program to solve for the natural frequencies and modes of vibration of a multi-degree-of-freedom mechanical system. The program should prompt the user for the

mass and stiffness matrices of the system, and then compute and print out the natural frequencies and modes of vibration.

8. Write a C++ program to simulate the behavior of a simple transformer. The program should prompt the user for the primary and secondary winding inductances and turns ratios, and then compute and print out the voltage and current ratios, as well as the phase angle shift, for a given input voltage.
9. Write a C++ program to calculate the magnetic field and flux density at a point due to a current-carrying wire. The program should prompt the user for the current, position, and orientation of the wire, and then compute and print out the magnetic field and flux density at a given point in space.
10. Write a C++ program to simulate the behavior of a simple three-phase AC system. The program should prompt the user for the line-to-line voltage, phase angle, and load impedance values, and then compute and print out the line current, phase current, and power factor for each phase.
11. Write a C++ program to calculate the power delivered by a transmission line. The program should prompt the user for the line voltage, current, and impedance values, and then compute and print out the active power, reactive power, and apparent power delivered by the line.
12. Write a C++ program to calculate the electric field and potential at a point in space due to a charged conductor. The program should prompt the user for the shape, size, and charge distribution of the conductor, and then compute and print out the electric field and potential at a given point in space.
13. Write a C++ program to simulate the behavior of a simple single-phase AC motor. The program should prompt the user for the resistance, inductance, and torque constants of the motor, and then compute and print out the current, speed, and power of the motor for a given applied voltage and frequency.
14. Write a C++ program to solve for the transfer function of a linear time-invariant system. The program should prompt the user for the system's state-space representation, and then compute and print out the transfer function.
15. Write a C++ program to simulate the behavior of a simple RC circuit. The program should prompt the user for the resistance and capacitance values of the circuit, and then

compute and print out the time constant, response to a step input, and impulse response of the circuit.

16. Write a C++ program to calculate the mutual inductance between two current-carrying wires. The program should prompt the user for the current, position, and orientation of the wires, and then compute and print out the mutual inductance.
17. Write a C++ program to simulate the behavior of a simple DC power system. The program should prompt the user for the voltage and current values at the source and load, and then compute and print out the power delivered and the efficiency of the system.
18. Write a C++ program to solve for the steady-state operating point of a nonlinear electrical network. The program should prompt the user for the network's topology and element values, and then compute and print out the node voltages and branch currents.
19. Write a C++ program to calculate the electric field and potential at a point in space due to a charged dielectric. The program should prompt the user for the dielectric's shape, size, permittivity, and charge distribution, and then compute and print out the electric field and potential at a given point in space.
20. Write a C++ program to simulate the behavior of a simple three-phase induction motor. The program should prompt the user for the resistance, inductance, and rotor speed of the motor, and then compute and print out the stator current, torque, and power factor for a given applied voltage and frequency.