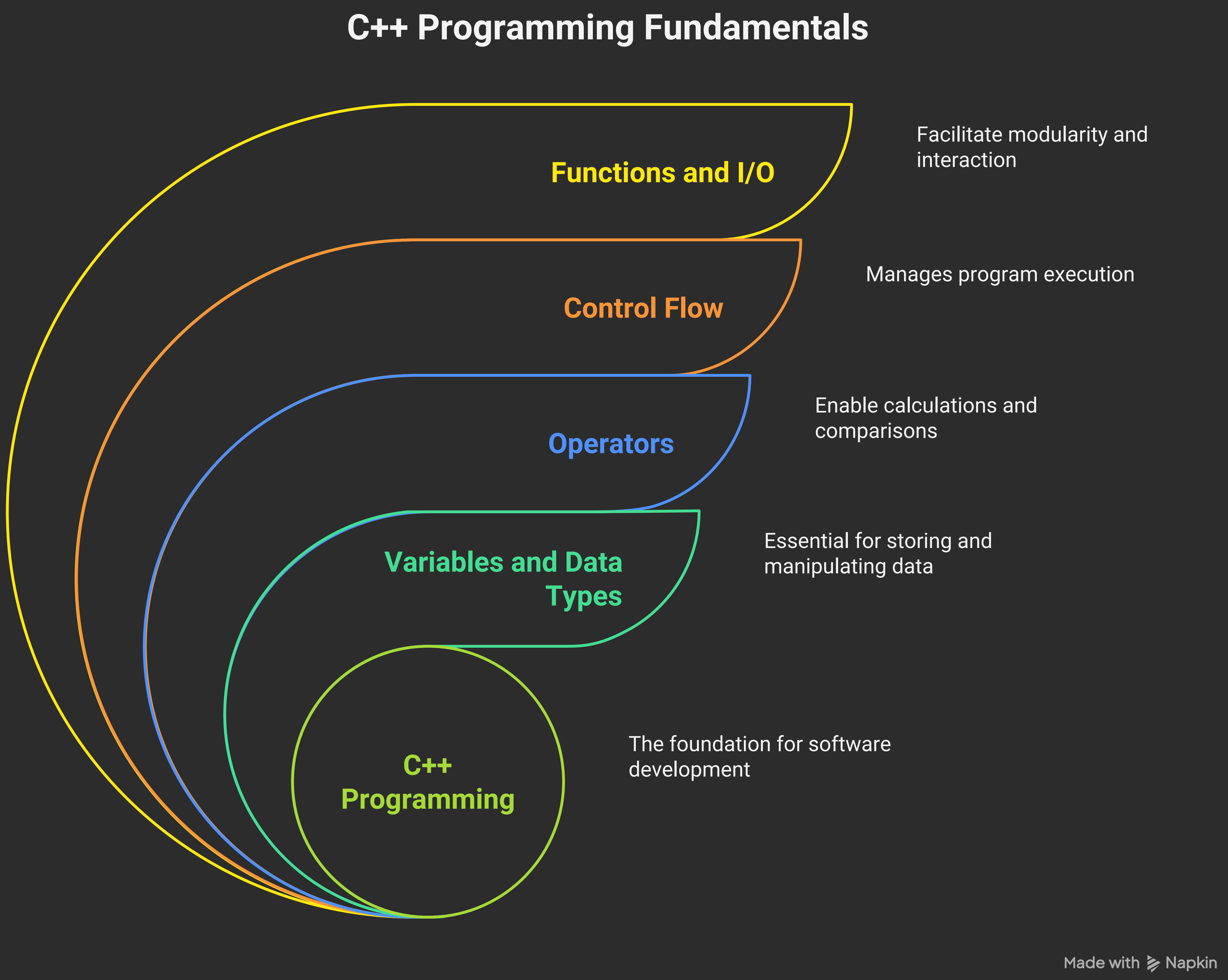
Beginner C++

This document provides a basic introduction to the C++ programming language, covering fundamental concepts such as variables, data types, operators, control flow, functions, and basic input/output. It is designed for individuals with little to no prior programming experience, aiming to equip them with the foundational knowledge necessary to start writing simple C++ programs.


C++ Programming Fundamentals
Functions and I/O
Facilitate modularity and interaction
Control Flow
Manages program execution
Operators
Enable calculations and comparisons
Variables and Data Types
Essential for storing and manipulating data
C++ Programming
The foundation for software development
Made with Napkin

Introduction to C++

C++ is a powerful, versatile, and widely-used programming language. It's an extension of the C language, adding object-oriented features like classes and objects. C++ is used in a wide range of applications, including operating systems, game development, high-performance computing, and embedded systems.

Setting up Your Environment

Before you can start writing C++ code, you need to set up a development environment. This typically involves installing a C++ compiler and a text editor or an Integrated Development Environment (IDE).

- **Compiler:** A compiler translates your C++ code into machine-readable instructions that your computer can execute. Popular C++ compilers include GCC (GNU Compiler Collection), Clang, and Microsoft Visual C++ Compiler.
- **Text Editor/IDE:** A text editor is a program that allows you to write and edit code. An IDE provides additional features like code completion, debugging tools, and build automation. Popular IDEs for C++ include Visual Studio Code (with C++ extension), Code::Blocks, and CLion.

Basic Syntax

A C++ program consists of a collection of statements. Each statement typically ends with a semicolon (;). C++ is case-sensitive, meaning that variableName and VariableName are treated as different identifiers.

Here's a simple C++ program:

```
#include <iostream>

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

- #include <iostream>: This line includes the iostream header file, which provides input/output functionalities.
- int main(): This is the main function, where the program execution begins.
- std::cout << "Hello, World!" << std::endl;: This line prints the text "Hello, World!" to the console. std::cout is the standard output stream, << is the insertion operator, and std::endl inserts a newline character.
- return 0;: This line indicates that the program has executed successfully.

Variables and Data Types

Variables are used to store data in a program. Each variable has a specific data type, which determines the kind of data it can hold.

Common Data Types

- int: Used to store integers (whole numbers).
- float: Used to store single-precision floating-point numbers (numbers with decimal points).
- double: Used to store double-precision floating-point numbers.
- char: Used to store single characters.
- bool: Used to store boolean values (true or false).

- std::string: Used to store sequences of characters (strings). Requires #include <string>.

Declaring Variables

To declare a variable, you specify its data type followed by its name:

```
int age;
float price;
char initial;
std::string name;
```

Initializing Variables

You can initialize a variable when you declare it:

```
int age = 30;
float price = 19.99;
char initial = 'A';
std::string name = "John Doe";
```

Operators

Operators are symbols that perform operations on variables and values.

Arithmetic Operators

- +: Addition
- -: Subtraction
- *: Multiplication
- /: Division
- %: Modulus (remainder)

Assignment Operators

- =: Assigns a value to a variable
- +=: Adds a value to a variable and assigns the result
- -=: Subtracts a value from a variable and assigns the result
- *=: Multiplies a variable by a value and assigns the result
- /=: Divides a variable by a value and assigns the result

Comparison Operators

- ==: Equal to
- !=: Not equal to
- >: Greater than
- <: Less than

- >=: Greater than or equal to
- <=: Less than or equal to

Logical Operators

- &&: Logical AND
- ||: Logical OR
- !: Logical NOT

Control Flow

Control flow statements allow you to control the order in which statements are executed.

`if` Statement

The if statement executes a block of code if a condition is true:

```
int age = 20;
if (age >= 18) {
  std::cout << "You are an adult." << std::endl;
}
```

`if-else` Statement

The if-else statement executes one block of code if a condition is true and another block of code if the condition is false:

```
int age = 16;
if (age >= 18) {
  std::cout << "You are an adult." << std::endl;
} else {
  std::cout << "You are a minor." << std::endl;
}
```

`else if` Statement

The else if statement allows you to check multiple conditions:

```
int score = 85;
if (score >= 90) {
  std::cout << "A" << std::endl;
} else if (score >= 80) {
  std::cout << "B" << std::endl;
} else if (score >= 70) {
  std::cout << "C" << std::endl;
} else {
  std::cout << "D" << std::endl;
}
```

`for` Loop

The for loop executes a block of code repeatedly for a specified number of times:

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

`while` Loop

The while loop executes a block of code repeatedly as long as a condition is true:

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

`do-while` Loop

The do-while loop executes a block of code at least once and then repeatedly as long as a condition is true:

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

Functions

Functions are reusable blocks of code that perform a specific task.

Defining Functions

To define a function, you specify its return type, name, and parameters:

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

- int: The return type of the function (the type of data the function returns).
- add: The name of the function.
- int a, int b: The parameters of the function (the inputs to the function).

Calling Functions

To call a function, you use its name followed by parentheses, passing any necessary arguments:

```
int sum = add(5, 3);
std::cout << sum << std::endl; // Output: 8
```

Input/Output

C++ provides input/output streams for interacting with the user.

- std::cout: The standard output stream (used for printing to the console).
- std::cin: The standard input stream (used for reading input from the console).

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

int main() {
  std::string name;
  std::cout << "Enter your name: ";
  std::getline(std::cin, name); // Use getline to read a line of text, including
spaces.
  std::cout << "Hello, " << name << "!" << std::endl;

  int age;
  std::cout << "Enter your age: ";
  std::cin >> age;
  std::cout << "You are " << age << " years old." << std::endl;

  return 0;
}
```

Conclusion

This document has provided a basic introduction to C++. It is important to practice writing code to solidify your understanding of these concepts. Further exploration of topics like object-oriented programming, data structures, and algorithms will build upon this foundation and allow you to create more complex and sophisticated programs.