# C programming notes quick revision
# 70-day challenge

## 1. Character Set

- **A character set is a collection of symbols that are used to represent text. In the context of programming languages, it refers to the set of characters that the language can recognize. For example, C uses the ASCII (American Standard Code for Information Interchange) character set.**
- **ASCII Table: ASCII is a 7-bit character encoding standard that represents text in computers. The ASCII table includes 128 characters (0-127) such as digits, letters, punctuation, and control characters.**

## 2. ASCII TABLE 🙂

| Decimal | Hex | Character | Description |
|---|---|---|---|
| 0 | 0 | NUL | Null character |
| 1 | 1 | SOH | Start of Header |
| 2 | 2 | STX | Start of Text |
| 3 | 3 | ETX | End of Text |
| 4 | 4 | EOT | End of Transmission |
| 5 | 5 | ENQ | Enquiry |
| 6 | 6 | ACK | Acknowledge |
| 7 | 7 | BEL | Bell |
| 8 | 8 | BS | Backspace |
| 9 | 9 | TAB | Horizontal Tab |
| 10 | 0A | LF | Line Feed |
| 11 | 0B | VT | Vertical Tab |
| 12 | 0C | FF | Form Feed |
| 13 | 0D | CR | Carriage Return |
| 14 | 0E | SO | Shift Out |
| 15 | 0F | SI | Shift In |
| 16 | 10 | DLE | Data Link Escape |
| 17 | 11 | DC1 | Device Control 1 |
| 18 | 12 | DC2 | Device Control 2 |
| 19 | 13 | DC3 | Device Control 3 |
| 20 | 14 | DC4 | Device Control 4 |
| 21 | 15 | NAK | Negative Acknowledge |
| 22 | 16 | SYN | Synchronous Idle |
| 23 | 17 | ETB | End of Block |

| 24 | 18 | CAN | Cancel |
|---|---|---|---|
| 25 | 19 | EM | End of Medium |
| 26 | 1A | SUB | Substitute |
| 27 | 1B | ESC | Escape |
| 28 | 1C | FS | File Separator |
| 29 | 1D | GS | Group Separator |
| 30 | 1E | RS | Record Separator |
| 31 | 1F | US | Unit Separator |
| 32 | 20 | (Space) | Space |
| 33 | 21 | ! | Exclamation mark |
| 34 | 22 | " | Double quote |
| 35 | 23 | # | Hash symbol |
| 36 | 24 | $ | Dollar sign |
| 37 | 25 | % | Percent |
| 38 | 26 | & | Ampersand |
| 39 | 27 | | Single quote |
| 40 | 28 | ( | Left parenthesis |
| 41 | 29 | ) | Right parenthesis |
| 42 | 2A | * | Asterisk |
| 43 | 2B | #ERROR! | Plus sign |
| 44 | 2C | , | Comma |
| 45 | 2D | - | Hyphen |
| 46 | 2E | . | Period |
| 47 | 2F | / | Slash |
| 48 | 30 | 0 | Zero |
| 49 | 31 | 1 | One |
| 50 | 32 | 2 | Two |
| 51 | 33 | 3 | Three |
| 52 | 34 | 4 | Four |
| 53 | 35 | 5 | Five |
| 54 | 36 | 6 | Six |
| 55 | 37 | 7 | Seven |
| 56 | 38 | 8 | Eight |
| 57 | 39 | 9 | Nine |
| 58 | 3A | : | Colon |
| 59 | 3B | ; | Semicolon |

# C programming notes quick revision
# 70-day challenge

| 60 | 3C | < | Less-than symbol |
|---|---|---|---|
| 61 | 3D | #ERROR! | Equal sign |
| 62 | 3E | > | Greater-than symbol |
| 63 | 3F | ? | Question mark |
| 64 | 40 | @ | At symbol |
| 65 | 41 | A | Uppercase A |
| 66 | 42 | B | Uppercase B |
| 67 | 43 | C | Uppercase C |
| 68 | 44 | D | Uppercase D |
| 69 | 45 | E | Uppercase E |
| 70 | 46 | F | Uppercase F |
| 71 | 47 | G | Uppercase G |
| 72 | 48 | H | Uppercase H |
| 73 | 49 | I | Uppercase I |
| 74 | 4A | J | Uppercase J |
| 75 | 4B | K | Uppercase K |
| 76 | 4C | L | Uppercase L |
| 77 | 4D | M | Uppercase M |
| 78 | 4E | N | Uppercase N |
| 79 | 4F | O | Uppercase O |
| 80 | 50 | P | Uppercase P |
| 81 | 51 | Q | Uppercase Q |
| 82 | 52 | R | Uppercase R |
| 83 | 53 | S | Uppercase S |
| 84 | 54 | T | Uppercase T |
| 85 | 55 | U | Uppercase U |
| 86 | 56 | V | Uppercase V |
| 87 | 57 | W | Uppercase W |
| 88 | 58 | X | Uppercase X |
| 89 | 59 | Y | Uppercase Y |
| 90 | 5A | Z | Uppercase Z |
| 91 | 5B | [ | Left square bracket |
| 92 | 5C | \ | Backslash |
| 93 | 5D | ] | Right square bracket |
| 94 | 5E | ^ | Caret (circumflex) |
| 95 | 5F | _ | Underscore |

| 96 | 60 | ` | Grave accent |
|---|---|---|---|
| 97 | 61 | a | Lowercase a |
| 98 | 62 | b | Lowercase b |
| 99 | 63 | c | Lowercase c |
| 100 | 64 | d | Lowercase d |
| 101 | 65 | e | Lowercase e |
| 102 | 66 | f | Lowercase f |
| 103 | 67 | g | Lowercase g |
| 104 | 68 | h | Lowercase h |
| 105 | 69 | i | Lowercase i |
| 106 | 6A | j | Lowercase j |
| 107 | 6B | k | Lowercase k |
| 108 | 6C | l | Lowercase l |
| 109 | 6D | m | Lowercase m |
| 110 | 6E | n | Lowercase n |
| 111 | 6F | o | Lowercase o |
| 112 | 70 | p | Lowercase p |
| 113 | 71 | q | Lowercase q |
| 114 | 72 | r | Lowercase r |
| 115 | 73 | s | Lowercase s |
| 116 | 74 | t | Lowercase t |
| 117 | 75 | u | Lowercase u |
| 118 | 76 | v | Lowercase v |
| 119 | 77 | w | Lowercase w |
| 120 | 78 | x | Lowercase x |
| 121 | 79 | y | Lowercase y |
| 122 | 7A | z | Lowercase z |
| 123 | 7B | { | Left curly brace |
| 124 | 7C | \| | Vertical bar (pipe) |
| 125 | 7D | } | Right curly brace |
| 126 | 7E | ~ | Tilde |
| 127 | 7F | DEL | Delete (Non-printable) |

## Key Points:

| |
|---|
| The first 32 characters (0-31) are control characters (non-printable), often used for text formatting and communication protocols. |
| Characters from 32-126 include printable characters, like numbers, letters, punctuation, and symbols. |
| The last character (127) is DEL, which is used to delete a character in text processing. |

### 3. Tokens

- In programming, a token is a basic unit of a program, such as a keyword, identifier, constant, or operator. The process of breaking a program into tokens is known as lexical analysis.
- Tokens can include:
  - Keywords: Reserved words that have special meaning (e.g., `if`, `else`, `while`).
  - Identifiers: Names used to identify variables, functions, etc.
  - Constants: Values that do not change.
  - Operators: Symbols like `+`, `-`, `*`, etc.
  - Punctuation: Symbols like `;`, `{`, `}`, etc.

# C programming notes quick revision
# 70-day challenge

## 4. Keywords

- **Keywords are reserved words in a programming language that cannot be used as identifiers (names for variables, functions, etc.). They are predefined by the language and have special meaning.**

| Keyword | Description | |
|---|---|---|
| auto | Used to define automatic variables (local variables by default). | |
| break | Exits from a loop or a switch statement. | |
| case | Marks a branch in a switch statement for specific values. | |
| char | Used to declare a variable of type char, which stores a single character. | |
| const | Specifies that the value of a variable cannot be changed. | |
| continue | Skips the current iteration of a loop and proceeds to the next iteration. | |
| default | Specifies the default case in a switch statement when no case matches. | |
| do | Used to start a do-while loop, which executes the loop body first and then checks the condition. | |
| double | Declares a variable of type double, used for floating-point numbers with double precision. | |
| else | Specifies the block of code to execute if the if condition is false. | |
| enum | Defines an enumerated data type, which represents a set of named integer constants. | |
| extern | Declares a variable or function that is defined outside the current file (external linkage). | |
| float | Declares a variable of type float, used for single-precision floating-point numbers. | |
| for | Used to start a for loop, which allows repeated execution of a block of code. | |
| goto | Transfers control to a labeled statement in the program. | |
| if | Used to start a conditional statement that executes a block of code if a condition is true. | |
| inline | Suggests to the compiler to insert the function's code at the point of the function call. | |
| int | Declares a variable of type int, used for integer values. | |
| long | Declares a variable of type long, used for long integers (larger range). | |
| register | Suggests that the variable be stored in a CPU register for faster access (though modern compilers often ignore this). | |
| return | Exits from a function and optionally returns a value to the caller. | |
| short | Declares a variable of type short, used for short integer values. | |
| signed | Specifies that a variable can store both positive and negative values (default for integers). | |
| sizeof | Returns the size (in bytes) of a data type or object. | |

# C programming notes quick revision
# 70-day challenge

| | | |
|---|---|---|
| static | Specifies that a variable has local scope but retains its value between function calls. Also used for limiting visibility of functions/variables within the file. | |
| struct | Defines a structure, which is a collection of different data types grouped together. | |
| switch | Starts a switch statement, which is used for multi-way branching based on the value of a variable. | |
| typedef | Creates a new name (alias) for an existing data type. | |
| union | Defines a union, which is a data structure where all members share the same memory location. | |
| unsigned | Specifies that a variable can only store non-negative values (i.e., no sign bit). | |
| void | Specifies a function that does not return a value or indicates that a pointer is of unknown type. | |
| volatile | Indicates that a variable's value can change unexpectedly, often used in hardware or signal handling. | |
| while | Starts a while loop, which executes the block of code as long as the condition is true. | |

**Notes:**

- **`auto`: By default, local variables in C are `auto` variables, meaning they are stored in the stack. The `auto` keyword is rarely used explicitly.**
- **`register`: This is a hint to the compiler to try to store the variable in a CPU register for faster access. However, modern compilers optimize this automatically, and its usage is less important today.**
- **`typedef`: This keyword is used to create aliases for data types. For example, `typedef unsigned long ulong;` allows you to use `ulong` instead of `unsigned long`.**
- **`volatile`: This keyword is essential when dealing with variables that can be modified outside the program flow (e.g., hardware registers or variables in interrupt routines).**

**C programming notes quick revision**
**70-day challenge**

## 4. Identifiers & Naming Rules

- Identifiers are names given to various program elements such as variables, functions, arrays, etc.
- Naming Rules for identifiers (specific to C or C-like languages) include:
  - An identifier must start with a letter (A-Z or a-z) or an underscore (`_`).
  - After the first character, it can contain letters, digits (0-9), or underscores.
  - It must not be a keyword.
  - C identifiers are case-sensitive (e.g., `Variable` and `variable` are different).
  - They cannot start with a number.

## 5. Constants

- Constants are values that do not change throughout the execution of a program.
- Types of Constants:
  - Integer constants: E.g., `100`, `-50`
  - Floating-point constants: E.g., `3.14`, `-0.001`
  - Character constants: E.g., `'A'`, `'b'`
  - String constants: E.g., `"Hello, World!"`
- Constants can be declared in C using `#define` (preprocessor directive) or `const` keyword
- Source code

```
#define PI 3.14  // Preprocessor constant

const int max_size = 100;  // Constant variable
```

## 6. Type Qualifiers

- Type qualifiers are keywords in C/C++ that modify the behavior or properties of data types.

**C programming notes quick revision**
**70-day challenge**

- **Common type qualifiers:**
  - **`const`: Specifies that a variable's value cannot be changed after it is initialized.**
  - **`volatile`: Specifies that the value of a variable can change unexpectedly, often used with hardware-level programming or when dealing with memory-mapped registers.**

  **Source code :**

  **const int x = 10;  // x cannot be modified**

  **volatile int flag;  // flag can be modified unexpectedly (e.g., hardware interrupt)**

**Variables in c programming**

**In C programming, variables are used to store data values that can be used and manipulated throughout the program. A variable is a container for storing data, and each variable has a type, which determines the kind of data it can store.**

**Here's an overview of how variables work in C:**

**1. Declaration of Variables**

Before using a variable in C, you must declare it by specifying its type and name. The general syntax is:

type variable_name;

For example:

int age;

float temperature;

char grade;

**2. Types of Variables**

- **Integer Types: Used to store whole numbers.**

- **`int`: Typically stores whole numbers (e.g., -5, 0, 10).**
- **`short`: A shorter version of `int`, uses less memory.**
- **`long`: A larger version of `int`, uses more memory.**
- **`long long`: Stores even larger numbers.**

- **Floating-Point Types: Used to store decimal numbers.**
  - **`float`: Stores single-precision floating-point numbers (e.g., 3.14).**
  - **`double`: Stores double-precision floating-point numbers for more accuracy.**
- **Character Type: Used to store single characters.**
  - **`char`: Stores a single character (e.g., 'a', 'B').**
- **Boolean Type (in C99 and later, but not in older versions of C):**
  - **`bool`: A type used to store `true` or `false` values.**

### 3. Initialization of Variables

**You can initialize a variable at the time of its declaration. For example:**

```
int age = 25;

float temperature = 98.6;

char grade = 'A';
```

**If a variable is not initialized, it will contain an undefined value (garbage value).**

### 4. Naming Rules for Variables

**In C, variables must follow specific rules when naming them:**

- **The name must start with a letter (`A-Z`, `a-z`) or an underscore (`_`).**
- **The rest of the name can include letters, numbers (`0-9`), and underscores.**
- **The name must not be a reserved keyword (e.g., `int`, `return`, `while`).**
- **Variable names are case-sensitive (e.g., `age` and `Age` are different variables).**
- **The name cannot exceed the limit set by the compiler (typically 31 characters).**

**C programming notes quick revision**
**70-day challenge**

### 5. Variable Scope

**The scope of a variable refers to the region of the program where it can be accessed:**

- **Local variables: Defined within a function or block, and can only be used within that function or block.**
- **Global variables: Defined outside all functions, usually at the top of the program, and can be accessed by any function.**

### 6. Types of Variable Storage Classes

**In C, the storage class determines the lifetime and visibility of a variable:**

- **Automatic (`auto`): The default storage class for local variables. These variables are created when the function is called and destroyed when the function exits.**
- **Static (`static`): Retains the value of the variable between function calls.**
- **Extern (`extern`): Declares a global variable that is defined outside the current file.**
- **Register (`register`): Suggests that the variable be stored in the CPU register for faster access (though it's not guaranteed).**

### 7. Constants

**A constant is a variable-like entity whose value cannot be changed after initialization. In C, constants are defined using the `const` keyword or the `#define` directive:**

**Using `const`:**

```
const int days_in_week = 7;
```

**Using `#define`:**

```
#define PI 3.14159
```

**Example Program:**

```
#include <stdio.h>

int main() {
```

```
    int age = 25;

    float temperature = 98.6;

    char grade = 'A';

    printf("Age: %d\n", age);

    printf("Temperature: %.2f\n", temperature);

    printf("Grade: %c\n", grade);

    return 0;

}
```

**8. Important Notes:**

- **Variables in C are strongly typed, meaning the type of data must be specified (e.g., `int`, `float`).**
- **C doesn't automatically check if a variable is initialized before use, which can lead to undefined behavior if you use uninitialized variables.**
- **Type casting: You can convert one data type to another (e.g., from `float` to `int`).**

---

**Operators :**

In C programming, operators are symbols used to perform operations on variables and values. Operators can be categorized into several types based on the kind of operations they perform.

# C programming notes quick revision
# 70-day challenge

## 1. Arithmetic Operators

| Operator | Description | Example |
|---|---|---|
| + | Addition | a + b |
| - | Subtraction | a - b |
| * | Multiplication | a * b |
| / | Division | a / b |
| % | Modulus (remainder) | a % b |

## 2. Relational operators

| Operator | Description | Example |
|---|---|---|
| #ERROR! | Equal to | a == b |
| != | Not equal to | a != b |
| > | Greater than | a > b |
| < | Less than | a < b |
| >= | Greater than or equal to | a >= b |
| <= | Less than or equal to | a <= b |

## 3. Logical operators
   a. Logical and &&
   b. Logical or ||
   c. Logical not !
## 4. Bitwise operators

| Operator | Description | Example |
|---|---|---|
| & | Bitwise AND | a & b |
| \| | \| | Bitwise OR |
| ^ | Bitwise XOR | a ^ b |
| ~ | Bitwise NOT (complement) | ~a |
| << | Left shift | a << 2 |
| >> | Right shift | a >> 2 |

**C programming notes quick revision**
**70-day challenge**

5. **Assignment operators**
    a. **Equal**
    b. **Add and assign ⇒ +=**
    c. **Sub and assign ⇒ -=**
    d. **Multiply and assign ⇒ *=**
    e. **Divide and assign ⇒ /=**
    f. **Modulus and assign ⇒ %=**
6. **Increment and decrement operators**
    a. **Post fix**
    b. **Pre fix**
7. **Conditional (ternary) operators**
    a. **For true ⇒ ?**
    b. **For false ⇒ :**
8. **Comma operators**
    a. **Comma ⇒ ,**
9. **Size of operators**
    a. **sizeof()**
10. **Pointer operators**
    a. **Address of ⇒ &**
    b. **Deference operators ⇒ ***
11. **Typecast operators**
    a. **(type)**

**Arithmetics operators exercise 50 questions**

**Basic Level (1-10)**

1. **Addition of two integers**
   **Write a program that takes two integers as input and calculates their sum.**

2. **Subtraction of two integers**
   **Write a program that takes two integers as input and calculates the**

**difference.**

3. **Multiplication of two integers**
   **Write a program that takes two integers as input and calculates their product.**

4. **Division of two integers**
   **Write a program that takes two integers as input and calculates the quotient.**

5. **Modulus operation of two integers**
   **Write a program that takes two integers as input and calculates the remainder.**

6. **Add and subtract two numbers**
   **Write a program that takes two integers as input and calculates the sum and difference of the numbers.**

7. **Multiply and divide two numbers**
   **Write a program that takes two integers as input and calculates their product and quotient.**

8. **Input two float numbers and perform arithmetic operations**
   **Write a program that takes two float numbers as input and performs addition, subtraction, multiplication, and division.**

9. **Perform addition and multiplication with user input**
   **Write a program that accepts two integers from the user, performs addition and multiplication, and displays the results.**

10. **Input a number and calculate its square**
    **Write a program that accepts an integer and calculates the square of that number.**

# C programming notes quick revision
# 70-day challenge

## Moderate Level (11-30)

11. **Average of two integers**
    Write a program that takes two integers as input and calculates their average.

12. **Find the difference and sum of two float numbers**
    Write a program that takes two float numbers and finds their sum and difference.

13. **Area of a rectangle**
    Write a program that takes the length and width of a rectangle and calculates its area.

14. **Area and perimeter of a circle**
    Write a program that accepts the radius of a circle and calculates its area and perimeter (circumference).

15. **Area of a triangle**
    Write a program that accepts the base and height of a triangle and calculates its area.

16. **Simple interest calculation**
    Write a program that accepts the principal amount, rate of interest, and time in years, and calculates the simple interest using the formula `SI = (P * R * T) / 100`.

17. **Temperature conversion (Celsius to Fahrenheit)**
    Write a program that takes the temperature in Celsius as input and converts it to Fahrenheit.

18. **Convert kilometers to miles**
    Write a program that takes a distance in kilometers and converts it to miles.

19. **Total cost after applying a discount**
    Write a program that takes the original price and discount percentage and

calculates the total price after the discount.

20. Perimeter of a rectangle
    Write a program that takes the length and width of a rectangle and calculates its perimeter.

21. Multiplication of a number with a constant factor
    Write a program that accepts a number and multiplies it by a constant factor (e.g., 5).

22. Swap two numbers using arithmetic operations
    Write a program that takes two integers as input and swaps their values using arithmetic operators.

23. Check if a number is even or odd
    Write a program that takes an integer as input and checks whether it is even or odd using the modulus operator.

24. Find the maximum of two numbers
    Write a program that takes two integers as input and finds the maximum of the two.

25. Find the minimum of two numbers
    Write a program that takes two integers as input and finds the minimum of the two.

26. Calculate the distance between two points
    Write a program that accepts the coordinates of two points `(x1, y1)` and `(x2, y2)` and calculates the distance between them using the distance formula:
    `sqrt((x2 - x1)^2 + (y2 - y1)^2)`.

27. Square root of a number
    Write a program that takes an integer and calculates its square root.

28. **Factorial of a number**
    Write a program that takes an integer as input and calculates its factorial.

29. **Check whether a number is positive, negative, or zero**
    Write a program that accepts a number as input and checks whether it is positive, negative, or zero.

---

**Advanced Level (31-50)**

31. **Calculate the compound interest**
    Write a program that takes the principal amount, rate of interest, time in years, and calculates the compound interest using the formula:
    `CI = P * (1 + R / 100)^T - P`.

32. **Perform a series of operations based on user choice**
    Write a program that asks the user to choose an operation (addition, subtraction, multiplication, or division) and perform that operation on two numbers.

33. **Volume of a cylinder**
    Write a program that accepts the radius and height of a cylinder and calculates its volume using the formula:
    `V = π * r^2 * h`.

34. **Check if a number is divisible by another**
    Write a program that takes two numbers as input and checks if the first number is divisible by the second.

35. **Number of digits in a number**
    Write a program that takes an integer as input and calculates how many digits it has.

---

# C programming notes quick revision
# 70-day challenge

shounaktiwari@gmail.com