C Language Notes

Beginner-Friendly Notes for Students, Exams, and Practice

Table of Contents

1. Introduction to C Language

What is C language?

C is a general-purpose, procedural programming language developed for writing system software and applications. It is considered the foundation of modern programming. Because it interacts closely with hardware while maintaining human-readable syntax, C is known as a middle-level language.

History of C programming

C was developed at Bell Laboratories in the early 1970s. It was created as an evolution of older languages like BCPL and B. The primary goal was to write the UNIX operating system.

Creator of C language

The C language was created by **Dennis Ritchie** in **1972**.

Importance of C programming

C is extremely important because many popular languages like C++, Java, and Python are based on C syntax. Learning C provides a deep understanding of how memory works, how programs execute, and how computers process instructions.

Applications of C language

- **Operating Systems:** Windows, Linux, and UNIX kernels are largely written in C.
- **Embedded Systems:** Microcontrollers in cars, microwaves, and watches use C.
- **Databases:** MySQL and Oracle database systems are built using C.
- **Compilers:** Many language compilers are written in C.
- **Gaming:** Core gaming engines often utilize C for high-speed graphics rendering.

> **Quick Revision:** C was created by Dennis Ritchie in 1972 at Bell Labs. It is a procedural, middle-level language used heavily for system programming.

2. Features of C Language

C is a highly versatile language due to its powerful features:

- **Simple:** C provides a structured approach, making it easy to learn and write. It has a rich set of built-in functions and operators.
- **Fast:** C programs execute much faster than interpreted languages like Python or Java because C is a compiled language that translates directly into machine code.
- **Portable:** C programs written on one computer can be run on another computer with little to no modification.
- **Structured programming language:** It allows breaking down a complex program into smaller, manageable functions, enhancing code readability and reusability.
- **Efficient:** C provides low-level memory access and pointer arithmetic, making it highly memory-efficient.

3. Structure of a C Program

Every C program follows a standard structure consisting of several sections.

Header files

Header files contain pre-defined standard library functions. They are included at the top using the `#include` directive. For example, `#include <stdio.h>` is required for input and output functions.

main() function

The `main()` function is the entry point of every C program. The execution of the program always starts from `main()` .

printf()

This is a standard library function used to print text or output to the console screen.

return statement

The `return 0;` statement inside the `main()` function indicates that the program has executed successfully without any errors.

4. First C Program

Hello World program

```
#include <stdio.h>

int main() {
    printf("Hello, World!");
return 0;
}
```

Explanation of each line

- `#include <stdio.h>`: Tells the compiler to include the Standard Input Output header file for using `printf()` .
- `int main()` : The main function where program execution begins. `int` means it will return an integer value.

  `{ ... }`

- : Curly braces define the start and end of the function body.
- `printf("Hello, World!");` : Prints the text "Hello, World!" to the screen. Every `;` statement in C ends with a semicolon ().
- : `return 0;` Returns 0 to the operating system, indicating successful execution.

Output example

```
Hello, World!
```

> **Exam Tip:** Never forget the semicolon `;` at the end of statements. Missing semicolons are the most common syntax errors for beginners!

5. Tokens in C

Tokens are the smallest individual units in a C program. The C compiler breaks the program into tokens before processing it.

Keywords

Keywords are reserved words that have special meaning to the compiler. You cannot use them as variable names. C has 32 standard keywords (e.g., `int`, `return`, `if`, `while`).

Identifiers

Identifiers are names given to variables, functions, arrays, etc. They are created by the programmer.

Constants

Constants are fixed values that do not change during program execution (e.g., , `'A'` `10` , `3.14`).

Variables

Variables are named memory locations used to store data that can be changed during execution.

Operators

Operators are symbols that perform mathematical or logical operations (e.g., `+`, `-`, `*`, `>`).

Special symbols

Symbols like `[ ]`, `( )`, `{ }`, `,`, `;` have specific syntactical meanings in C.

6. Data Types in C

Data types specify the type of data that a variable can store. The fundamental data types are:

Data Type	Description	Size (typical)	Format Specifier
`Int`	**Used to store whole numbers.**	**4 bytes**	`%d`
`float`	**Used to store decimal/fractional numbers.**	**4 bytes**	`%f`
`char`	**Used to store a single character.**	**1 byte**	`%c`
`double`	**Used for high-precision decimal numbers.**	**8 bytes**	`%lf`
`void`	**Represents "no value" or "empty".**	**-**	

7. Variables and Constants

Declaration and initialization

Declaration tells the compiler about the variable's name and type. **Initialization** means assigning a starting value.

```
int age;       // Declaration age =
20;      // Initialization

float salary = 5000.50; // Declaration and initialization together
```

Naming rules

- Must begin with a letter or underscore `_` ().
- Cannot contain spaces or special characters (except `_`).
- Cannot be a keyword.
- C is case-sensitive (`Age` and `age` are different).

Examples

```
const float PI = 3.14; // Constant declaration using const keyword
```

8. Input and Output Functions

printf()

Used to display output on the screen.

```
printf("Age is %d", age);
```

scanf()

Used to take input from the user via the keyboard. It requires the address of the variable, represented by the `&` symbol.

```
scanf("%d", &age);
```

Format specifiers

Format specifiers tell `printf` and `scanf` what type of data to read or print (e.g., `%d` for int, `%f` for float, `%c` for char).

9. Operators in C

Arithmetic operators

Perform mathematical operations: `+` (Addition), `-` (Subtraction), `*` (Multiplication), `/` (Division), `%` (Modulus/Remainder).

Relational operators

Compare two values: `==` (Equal to), `!=` (Not equal), `<`, `>`, `<=`, `>=`. They return 1 (true) or 0 (false).

Logical operators

Used to combine conditions: `&&` (Logical AND), `||` (Logical OR), `!` (Logical NOT).

Assignment operators

Assign values to variables: `=`, `+=`, `-=`, `*=`, `/=`.

Increment and decrement operators

Increase or decrease a value by 1: `++` (Increment), `--` (Decrement).

10. Decision Making Statements

Decision making statements decide the flow of program execution based on conditions.

if statement

Executes a block of code only if the condition is true.

```
if (age >= 18) {
printf("You can vote.");
}
```

if else statement

Provides an alternative block of code if the condition is false.

```
if (age >= 18) {
printf("Adult");
} else {     printf("Minor");
}
```

nested if

`if` statement inside another `if` An statement.

switch statement

An alternative to multiple `if-else` conditions. It compares a variable against multiple cases.

```
int choice = 2;
switch(choice) {
case 1:

printf("One");
break;     case 2:

printf("Two");
break;     default:
        printf("Invalid");
}
```

11. Loops in C

Loops are used to execute a block of code repeatedly until a condition is met.

for loop

Used when the number of iterations is known beforehand.

```
for (int i = 1; i <= 5; i++) {
printf("%d ", i);
}
```

while loop

Used when the number of iterations is not known, and checking occurs before execution.

```
int i = 1;
while (i <= 5) {
printf("%d ", i);     i++;
}
```

do while loop

Similar to the `while` loop, but it guarantees that the loop runs at least once because the condition is checked at the end.

```
int i = 1;
do {
printf("%d ", i);
i++;
} while (i <= 5);
```

12. Arrays in C

An array is a collection of variables of the same data type stored in contiguous memory locations.

One-dimensional arrays

The simplest form of an array, acting like a list of items.

Array declaration

```
int numbers[5]; // Declares an integer array of size 5
```

Array examples

```
int marks[3] = {85, 90, 78};
printf("First subject marks: %d", marks[0]); //
Array index starts at 0
```

13. Strings in C

C does not have a built-in string data type. A string is represented as a one-dimensional array of characters terminated by a null character `'\0'` .

String declaration

```
char name[20] = "John";
```

String functions

Found in `<string.h>` :

- `strlen(str)` : Returns the length of the string.
- `strcpy(dest, src)` : Copies source string to destination.
- `strcmp(str1, str2)` : Compares two strings.
- `strcat(str1, str2)` : Concatenates (joins) two strings.

14. Functions in C

A function is a block of code that performs a specific task. It promotes code reusability.

Function declaration

Tells the compiler about the function name, return type, and parameters.

```
int add(int a, int b);
```

Function definition

The actual body of the function.

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

Function calling

Executing the function from `main()` or another function.

```
int sum = add(5, 10);
```

15. Pointers Basics

A pointer is a variable that stores the memory address of another variable.

Introduction to pointers

Pointers provide direct memory access, making programs powerful but requiring careful handling.

```
int var = 10; int *ptr;
ptr = &var; // ptr now stores the address of var
```

Pointer declaration

Use the `*` operator to declare a pointer.

Pointer examples

```
printf("Value of var: %d\n", var);
printf("Address of var: %p\n", &var);
printf("Value stored in ptr: %p\n", ptr);
printf("Value pointed to by ptr: %d\n", *ptr); // Dereferencing
```

16. File Handling in C

File handling allows programs to read from and write to external files.

fopen()

Opens a file. Modes include `"r"` (read), `"w"` (write), `"a"` (append).

fclose()

Closes the opened file to release resources.

fprintf() and fscanf()

Used to write to and read from files, similar to `printf` and `scanf`.

```
FILE *file = fopen("data.txt",
"w"); if (file != NULL) {
fprintf(file, "Hello File!");
fclose(file);
}
```

17. Practical C Programs

---

1. Addition of two numbers

```
#include <stdio.h> int main() {
int a = 5, b = 10;
printf("Sum = %d", a + b);
return 0;
}
```

2. Even or odd

```
#include
<stdio.h> int
main() {     int
num = 7;
    if(num % 2 == 0)
printf("Even");     else
printf("Odd");     return 0;
}
```

3. Prime number

```
#include <stdio.h>
int main() {
    int n = 7, i, flag = 0;
for(i = 2; i <= n/2; ++i) {
if(n % i == 0) {
flag = 1; break;
        }
}
    if(flag == 0 && n > 1)
printf("Prime");     else printf("Not
Prime");     return 0;
}
```

4. Factorial

```
#include <stdio.h>
int main() {
    int n = 5, i, fact = 1;
    for(i = 1; i <= n; i++) fact *= i;
printf("Factorial = %d", fact);
return 0;
}
```

5. Fibonacci series

```
#include <stdio.h>
int main() {
    int n = 5, t1 = 0, t2 = 1,
nextTerm;     for (int i = 1; i <= n;
++i) {         printf("%d ", t1);
nextTerm = t1 + t2;         t1 = t2;
t2 = nextTerm;
    }
return 0;
}
```

6. Reverse number

```
#include <stdio.h>
int main() {
    int n = 123, rev = 0,
remainder;     while (n != 0) {
remainder = n % 10;         rev =
rev * 10 + remainder;         n /=
10;
    }
    printf("Reversed = %d", rev);
return 0;
}
```

7. Largest of three numbers

```
#include <stdio.h>
int main() {
    int a = 10, b = 20, c = 15;
    if(a >= b && a >= c) printf("%d is largest", a);
else if(b >= a && b >= c) printf("%d is largest", b);
else printf("%d is largest", c);     return 0;
}
```

8. Swap two numbers (without third variable)

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

9. Check leap year

```
#include <stdio.h>
int main() {
int year = 2024;
    if ((year % 4 == 0 && year % 100 != 0) || (year % 400 ==
0))         printf("Leap Year");     else
        printf("Not a Leap Year");
return 0;
}
```

10. Sum of natural numbers

```
#include <stdio.h>
int main() {
    int n = 10, sum = 0;
    for(int i = 1; i <= n; i++) sum +=
i;     printf("Sum = %d", sum);
return 0;
}
```

11. Palindrome number

```
#include <stdio.h>
int main() {
    int n = 121, reversed = 0, remainder,
original;     original = n;     while (n != 0) {
remainder = n % 10;
        reversed = reversed * 10 + remainder;
n /= 10;
    }
    if (original == reversed)
printf("Palindrome");     else printf("Not
Palindrome");     return 0;
}
```

12. Armstrong number

```
#include <stdio.h>
int main() {
    int num = 153, original, rem, result =
0;     original = num;     while (original
!= 0) {         rem = original % 10;
result += rem * rem * rem;         original
/= 10;
    }
    if (result == num)
printf("Armstrong");     else printf("Not
Armstrong");     return 0;
}
```

13. Simple interest

```
#include <stdio.h>
int main() {
    float p = 1000, r = 5, t = 2, si;
si = (p * r * t) / 100;
    printf("Simple Interest = %.2f", si);
return 0;
}
```

14. Area of circle

```
#include <stdio.h>
int main() {
    float r = 5.0, area;
area = 3.14159 * r * r;
printf("Area = %.2f", area);
return 0;
}
```

15. Multiplication table

```
#include
<stdio.h> int
main() {     int
n = 5;
    for (int i = 1; i <= 10; ++i) {
        printf("%d * %d = %d\n", n, i, n * i);
    }
return 0;
}
```

16. Count digits

```
#include <stdio.h>
int main() {
    long long n =
12345;     int count =
0;     while (n != 0) {
n /= 10;
++count;
    }
    printf("Number of digits: %d", count);
return 0;
}
```

17. Sum of digits

```
#include <stdio.h>
int main() {
    int n = 123, sum =
0, m;     while(n > 0) {
m = n % 10;          sum
= sum + m;          n = n
/ 10;
    }
    printf("Sum = %d", sum);
return 0;
}
```

18. Linear search in array

```
#include <stdio.h>
int main() {
    int arr[] = {10, 20, 30, 40,
50};     int key = 30, n = 5, found
= 0;     for(int i=0; i<n; i++) {
if(arr[i] == key) {
            printf("Found at index %d",
i);             found = 1;
break;
        }
}
    if(!found) printf("Not found");
return 0;
}
```

19. String length without strlen

```
#include <stdio.h>
int main() {
    char str[] = "Hello";
int length = 0;
    while(str[length] != '\0') {
length++;
    }
    printf("Length = %d", length);
return 0;
}
```

20. Function to add two numbers

```
#include <stdio.h> int
add(int a, int b) {
return a + b;
}
int main() {
    int result = add(15, 25);
printf("Result = %d", result);
return 0;
}
```

18. Common Errors in C

- **Missing semicolon (;):**Every statement must end with a semicolon. Forgetting it causes syntax errors.
- **Syntax errors:** Typos in keywords, missing curly braces `{}`, or missing parentheses `()`.
- **Variable declaration mistakes:** Using a variable before declaring it or changing a `const` variable.
- **Wrong format specifiers:** Using `%d` for floats or missing the `&` symbol in `scanf`.
- **Array Out of Bounds:** Accessing an index that is greater than the array size (e.g., `arr[5]` in a 5-element array since max index is 4).

19. Viva Questions and Answers

1. **Who is the father of C language?** Dennis Ritchie.
2. **What is the extension of a C file?** `.c`
3. **What is a compiler?** A program that translates C code into machine code.
4. **What is the use of main() function?** It is the starting point of program execution.
5. **What are keywords?** Reserved words with special meanings. (C has 32).
6. **Difference between = and ==?** `=` is an assignment operator, `==` is a relational operator (comparison).
7. **What is the size of an int data type?** Usually 4 bytes on modern compilers.
8. **What is a pointer?** A variable that holds the memory address of another variable.
9. **What is an array?** A collection of similar data type items stored in contiguous memory.
10. **What is a string in C?** An array of characters ending with a null character `'\0'`.
11. **What does #include do?** It includes the contents of a header file into the program.
12. **Why do we use & in scanf?** It denotes the memory address where input data should be stored.
13. **What is an infinite loop?** A loop whose condition never becomes false.

14. **Difference between break and continue?** `break` exits the loop completely, `continue` skips the current iteration and proceeds to the next.
15. **What is a preprocessor directive?** Commands processed before actual compilation, starting with . `#`
16. **What is the return type of main by default?** `int` .
17. **Can we use switch with float values?** No, only with integers and characters.
18. **What is an algorithm?** A step-by-step logical procedure to solve a problem.
19. **What is NULL pointer?** A pointer that does not point to any valid memory location.
20. **What is local vs global variable?** Local is inside a block/function, global is outside all functions and accessible anywhere.

20. MCQs and Practice Questions

Multiple Choice Questions

1. **Which of the following is a valid C variable name?**
   a) 1st_name b) _name c) first-name d) int*Answer: b*
2. **Which data type is used to store characters?**
   a) int b) float c) char d) double*Answer: c*
3. **Which operator is used for logical AND?**
   a) & b) && c) | d) ||
   Answer: b
4. **What will % operator return?**
   a) Quotient b) Remainder c) Percentage d) None*Answer: b*
5. **Array indexing starts from:**
   a) 1 b) 0 c) -1 d) User defined*Answer: b*
6. **Which function is used to read input?**
   a) printf b) read c) input d) scanf*Answer: d*

7. **What is the size of char in C?**
   a) 1 byte b) 2 bytes c) 4 bytes d) 8 bytes*Answer: a*
8. **Which keyword is used to stop loop execution?**
   a) stop b) halt c) break d) continue*Answer: c*
9. **A pointer stores:**
   a) Value b) Address c) String d) Array
   Answer: b
10. **Which of these is not a loop?**
    a) for b) while c) switch d) do-while*Answer: c*
11. **Which header file is required for `printf()` and `scanf()` ?**
    a) <math.h> b) <string.h> c) <stdio.h> d) <conio.h>*Answer: c*
12. **What is the correct symbol used to end a statement in C?**
    a) : b) ; c) . d) ,
    Answer: b
13. **Which loop executes at least one time even if the condition is false?**
    a) for b) while c) do-while d) switch*Answer: c*
14. **Which operator is used to access the value at an address through a pointer?**
    a) & b) * c) % d) # *Answer: b*
15. **Which keyword is used to define a constant variable?**
    a) static b) const c) final d) define
    Answer: b
16. **Which function is used to open a file in C?**
    a) open() b) fileopen() c) fopen() d) newfile()*Answer: c*
17. **What is the output data type of the expression `5 / 2` in basic integer division?** a) 2.5 b) 2 c) 3 d) 2.0
    Answer: b
18. **Which function is used to close a file?**
    a) endfile() b) fileclose() c) fclose() d) closefile()*Answer: c*
19. **Which symbol is used to get the address of a variable?**

a) * b) & c) % d) *@Answer: b*

20. **Which of the following is a decision-making statement?**
    a) if b) printf c) scanf d) include
    Answer: a

Short Answer Practice Questions

1. Write a program to check if a number is positive, negative, or zero.
2. Explain the difference `while` and `do-while` between loops.
3. Write a function to calculate the square of a number.
4. Explain call by value vs call by reference.
5. Write a program to print elements of an array.
6. Define variable and constant with one example each.
7. What is the role of the `main()` function in C?
8. Explain any four features of the C language.
9. Write the syntax of `switch` the statement.
10. Differentiate between an array and a string in C.
11. What is a pointer? Why is it useful?
12. Write short notes `fopen()` and `fclose()` on .
13. List the basic data types in C.
14. What are format specifiers? Give any four examples.
15. Explain the use `return 0;` of in a C program.

21. Revision Notes

Important formulas and syntax

- **If-else:** `if (condition) { } else { }`
- **For loop:** `for (initialization; condition; increment) { }`
- **Array declaration:**
```
data_type
array_name[size];
```
- **Pointer declaration:**
```
data_type
*pointer_name;
```

Exam-day checklist

- Always check for semicolons at the end of statements.
- Remember `&` in `scanf` statements.
- Check array boundaries to prevent out-of-bounds logic errors.
- Format specifiers: `%d` (int), `%f` (float), `%c` (char), `%s` (string).

> **Final Exam Tip:** Write clean code, indent your loops and conditional blocks properly, and trace your logic manually with rough values before finalizing your answer! Good Luck!