C Language with explanations in Hindi

Part 1

1. Introduction to C Language (सी लैंग्वेज का परिचय)

C Language ek high-level programming language hai jo 1970s mein Dennis Ritchie ne develop ki thi. Yeh language bahut powerful hai aur system programming (jaise operating systems banane) ke liye widely use hoti hai. C language ko low-level hardware operations ko control karne ke liye design kiya gaya tha, lekin yeh high-level features bhi provide karti hai, jaise variables, loops, functions, etc.

Hinglish Explanation:

"C language ek programming language hai jo easy aur efficient code likhne ke liye use hoti hai. Iska main feature hai ki yeh directly hardware ke saath interact kar sakti hai, matlab jo computer ka internal system hai, usse closely kaam karti hai. Agar aapko operating system ya embedded systems banana ho, toh C language sabse zyada use hoti hai. C ki syntax simple hai, lekin yeh power-packed aur flexible bhi hai."

C Introduction based questions

Q. What is C?

"C ek general-purpose programming language hai, jo Dennis Ritchie ne 1972 mein Bell Laboratories mein banayi thi. Yeh ek bahut hi popular language hai, despite being old. Iski popularity ka main reason yeh hai ki yeh computer science ke field mein ek fundamental

language hai. C ka UNIX ke saath bhi ek strong connection hai, kyunki yeh UNIX operating system ko likhne ke liye develop ki gayi thi."

Q. Why Learn C?

"C duniya ke sabse popular programming languages mein se ek hai. Agar aapko C aata hai, toh aapko doosri popular languages jaise Java, Python, C++, C#, etc. seekhne mein koi problem nahi hogi, kyunki unka syntax C se kaafi similar hota hai. Agar aapko C aata hai, toh aapko samajh aayega ki computer memory kaise kaam karti hai. C bahut fast hai, doosri programming languages jaise Java aur Python ke comparison mein. Aur sabse acchi baat yeh hai ki C bohot versatile hai; yeh dono applications aur technologies mein use kiya jaa sakta hai."

Difference between C and C++

"C++ ko C ka extension banake develop kiya gaya tha, aur dono languages ka syntax almost same hota hai. C aur C++ ke beech ka main difference yeh hai ki C++ mein classes aur objects ka support hota hai, jabki C mein yeh cheez nahi hoti."

2. Data Types (डेटा प्रकार)

"C mein different types ke data types hote hain. Yeh data types alag-alag values ko store karne ke liye use hote hain."

- **Integer (int): "Integer type ke data mein sirf whole numbers, yaani ki poornank, hote hain."**

Example:

- int a = 5;

- **Floating Point (float, double): "Inka use decimal values ko store karne ke liye kiya jata hai."**
- **Example:**
  - **float x = 3.14;**
  - **double y = 3.14159;**

- **Character (char): "Yeh ek single character ko store karta hai."**
- **Example:**
  - **char c = 'A';**
- **Void: "Yeh koi value return nahi karta hai, yaani iska koi data type nahi hota."**

  Example:

  - **void function Name()**

3. Variables (चर)

"Variable ek naam hota hai jise hum data store karne ke liye use karte hain. C language mein, kisi bhi variable ko declare karne se pehle yeh specify karna padta hai ki uska data type kya hoga."

Example:
int a;   // Integer variable
float b;   // Floating-point variable
char c;   // Character variable

4. Operators (संचालक)

"C language mein operators ka use data ko manipulate karne ke liye kiya jata hai. Kuch common operators hain:"

- **Arithmetic Operators (+, -, *, /, %): "Inka use ganana (calculation) karne ke liye kiya jata hai."**
  - **Example: int result = a + b;**
- **Relational Operators (==, !=, >, <, >=, <=):**

  "Inka use do values ke beech comparison karne ke liye kiya jata hai."
  - **Example: if (a == b)**
- **Logical Operators (&&, ||, !): "Inka use logical operations ke liye kiya jata hai."**
  - **Example: if (a > 0 && b > 0)**
- **Assignment Operator (=): "Iska use variable ko value assign karne ke liye kiya jata hai."**
  - **Example: int a = 10;**

Let's create our first C file

**"Dev C/C++ kholen aur File > New > Empty File par jayein.
Nimn C code likhein aur file ko 'myfirstprogram.c' ke roop mein save karein (File > Save As) ."**

Input:

myfirstprogram.c (file name)

```
#include <stdio.h>
int main() {
  printf("Hello World!");
  return 0;
}
```

Then, go to **Build > Build and Run** to run (execute) the program. The result will look something to this:

Output :

```
Hello World!
Process returned 0 (0x0) execution time : 0.011 s
Press any key to continue.
```

C Statements

Statements

Ek computer program computer dwara 'execute' kiye jaane wale 'instructions' ki ek list hai.

Ek programming language mein, in programming instructions ko 'statements' kaha jata hai.

Nimn kathin 'Hello World' text ko screen par print karne ke liye compiler ko 'instruction' deta hai.

1. Expression Statement (व्यक्तिकरण कथन):

यह वह स्टेटमेंट है जो किसी अभिव्यक्ति को निष्पादित करता है, जैसे:

```
x = 5;
```

2. Selection Statement (चयन कथन):

Is statement ka use decision lene ke liye kiya jata hai. Jaise if aur switch:

```
if (x > 0)

   {
   printf("Positive number");
   }
```

3. Iteration Statement (आवृत्ति कथन):

Inka use loop banane ke liye kiya jata hai, jaise for, while, aur do-while.

```
  for (int i = 0; i < 5; i++)

   {
        printf("%d\n", i);
  }
```

4. Jump Statement (कूदने वाला कथन):

Inka use program ke kisi jagah se doosri jagah jump karne ke liye kiya jata hai, jaise break, continue, goto, aur return:

```
if (x == 0) {
     return;
  }
```

5. Declaration Statement (घोषणा कथन):

Yeh statement kisi variable ya function ko declare karne ke liye use hota hai:

int x;

In sab statements ka use C program mein different tasks ko perform karne ke liye kiya jata hai.

Example:

```
printf("Hello World!");
```

Important hai ki aap statement ke end mein semicolon ; daalein.

Agar aap semicolon ; bhool jaate hain, toh error aa sakti hai aur program run nahi karega:

Example

```
printf("Hello World!")
 expected ';' before 'return'
```

Many Statements

Zyaadatar C programs mein kai statements hote hain.
Ye statements ek ke baad ek execute hote hain, jaise woh likhe gaye hain:

Example

```
printf("Hello World!");
printf("Have a good day!");
return 0;
```

Example explained

Is example mein humein 3 statements milte hain:

1. printf("Hello World!");
2. printf("Have a good day!");
3. return 0;

Pehla statement pehle execute hoga (screen par "Hello World!" print hoga).
Phir doosra statement execute hoga (screen par "Have a good day!" print hoga).
Aur aakhri mein teesra statement execute hoga (C program successfully end hoga).

C Output (Print Text)

Output (Print Text)

C mein values ya text output karne ke liye printf() function ka use kiya jata hai:

Example

```
#include <stdio.h>

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

Double Quotes

Jab aap text ka use kar rahe ho, toh woh hamesha double quotation marks "" mein hona chahiye.
Agar aap double quotes bhool jaate hain, toh error aa sakti hai:

Example

```
printf("This sentence will work!");

printf(This sentence will produce an error.);
```

Many `printf` Functions

Aap jitni chahein printf() functions ka use kar sakte hain. Lekin, yeh dhyaan rakhein ki yeh output ke end mein new line insert nahi karta:

Example

```
#include <stdio.h>

int main() {
  printf("Hello World!");
  printf("I am learning C.");
  printf("And it is awesome!");
  return 0;
```

C Variables

Variables

Variables wo containers hote hain jo data values ko store karte hain, jaise numbers aur characters.
C mein alag-alag types ke variables hote hain (jo alag keywords se define kiye jaate hain), for example:

- int - integers (whole numbers), bina decimals ke, jaise 123 ya -123
- float - floating point numbers, decimals ke saath, jaise 19.99 ya -19.99
- char - single characters, jaise 'a' ya 'B'. Characters single quotes ' mein hote hain.

Declaring (Creating) Variables

Variable banane ke liye, uska type specify karo aur value assign karo:

Syntax

type variableName = value;

Jahan type C ke kisi type ka hota hai (jaise int), aur variableName variable ka naam hota hai (jaise x ya myName). Equal sign = ka use value assign karne ke liye hota hai.
Toh, agar aapko koi number store karne waala variable banana ho, toh dekhiye yeh example:

Example

Ek variable banayein jiska naam myNum ho, type int ho aur value 15 assign karein:

```
int myNum = 15;
```

Aap variable declare karte waqt value assign na karen, aur baad mein value assign kar sakte hain:

Example

```
// Declare a variable
int myNum;

// Assign a value to the variable
myNum = 15;
```

Output Variables

You learned from the output chapter that you can output values/print text with the **printf()** function:

Example

```
printf("Hello World!");
```

In many other programming languages (like Python, Java, and C++), you would normally use a print function to display the value of a variable. However, this is not possible in C:

Example

```
int myNum = 15;
printf(myNum);  // Nothing happens
```

C language mein Variable (चर) kya hota hai?

Variable ek naam hota hai jo program mein kisi data (value) ko store (sangrahit) karne ke liye use kiya jata hai. Yeh computer ki memory mein ek jagah ko refer karta hai, jahan koi value rakhi jaati hai. Aap us value ko baad mein use ya change kar sakte hain.

◆ Example:

```
int age = 25;
```

Yahan:
- int = Data Type, jo batata hai ki yeh ek integer (poorna ank) hai.
- age = Variable ka naam.
- 25 = Wo value jo is variable mein rakhi gayi hai.

◆ Variable कैसे बनाते हैं? (Syntax):

```
<data_type> variable name = value;
```

Example:

```
float price = 99.50;
char grade = 'A';
```

◈ कुछ सामान्य Data Types:

Data Type	Hindi Arth	Udaharan
int	Poornank sankhya	int count = 10;
float	Dashmalav sankhya	float pi = 3.14;
char	Ek akshar	char ch = 'A';
double	Bada dashmalav maan	double d = 99.9999;

◆ Variable ke naam rakhne ke niyam (Rules for Naming):

1. Naam mein keval akshar (A-Z/a-z), ank (0-9) aur _ (underscore) ho sakte hain.
2. Naam sankhya se shuru nahi ho sakta.
3. Koi keyword (jaise int, return, if) variable ka naam nahi ho sakta.
4. Naam sensitive hote hain — Age aur age alag-alag hain.

◆ Variable ke use:

```
#include <stdio.h>

int main() {
    int a = 10;
    int b = 20;
    int sum = a + b;
    printf("Plus = %d", sum);
    return 0;
}
```

Output:

```
Plus = 30
```

C Data Types

Data Types

📌 Data Type (डेटा टाइप) क्या है?

Data Type yeh determine karta hai ki kisi variable (वेरिएबल) mein kaun sa prakar ka data (jaise ki sankhya, akshar, dashmalav aadi) sangrahit (store) kiya ja sakta hai.
C bhasha mein, har variable ko declare karte waqt humein yeh batana hota hai ki usmein kaun sa data store hoga — isse hum data type kehte hain.

◈ मुख्य प्रकार के Data Types:

Data Type	उपयोग (Use)	उदाहरण (Example)
int	पूर्णांक (Integer)	int age = 25;
float	दशमलव संख्या (Decimal Number)	float pi = 3.14;
double	बड़ी दशमलव संख्या	double d = 3.14159265;
char	एक अक्षर (Single Character)	char grade = 'A';

◈ 1. int (इन्टिजर)

Poora ank (whole number) store karta hai. Dashmalav nahi hota.

int x = 10;

◈ 2. float (फ्लोट)

Dashmalav sankhya (decimal number) store karta hai. 4 byte memory leta hai.

float y = 3.14;

◈ 3. double (डबल)

float se zyada sathikta (precision) deta hai. 8 byte memory leta hai.

double z = 3.1415926535;

◈ 4. char (कैरेक्टर)

Sirf ek akshar store karta hai. Usse ' ' (single quotes) mein likha jata hai.

char ch = 'A';

✓ More Information:

Data Type	Size (आकार)	Range (सीमा)
int	4 bytes	-2,147,483,648 to 2,147,483,647
float	4 bytes	~±3.4e-38 to ±3.4e+38
double	8 bytes	~±1.7e-308 to ±1.7e+308
char	1 byte	-128 to 127 (ASCII value)

Jaise Variables chapter mein samjhaya gaya hai, C mein variable ka ek specified data type hona chahiye, aur aapko usse display karne ke liye printf() function mein format specifier ka use karna padta hai:

Example

```
// Create variables

int myNum = 5;          // Integer (whole number)

float myFloatNum = 5.99;   // Floating point number

char myLetter = 'D';      // Character


// Print variables

printf("%d\n", myNum);

printf("%f\n", myFloatNum);

printf("%c\n", myLetter);
```

Basic Data Types

Data type yeh specify karta hai ki variable kis size aur type ki information ko store karega.
Is tutorial mein, hum sabse basic data types pe focus karenge:

Data Type	Size	Description	Example
int	2 ya 4 bytes	Whole numbers, bina decimals ke.	1
float	4 bytes	Fractional numbers, jo ek ya zyada decimals rakhte hain. 6-7 decimal digits tak store karne ke liye kaafi hai.	1.99
double	8 bytes	Fractional numbers, jo ek ya zyada decimals rakhte hain. 15 decimal digits tak store karne ke liye kaafi hai.	1.99
char	1 byte	Single character/letter/number, ya ASCII values.	'A'

Basic Format Specifiers

Har data type ke liye alag format specifiers hote hain. Yeh kuch common hain:

Format Specifier	Data Type
%d ya %i	int
%f ya %F	float
%lf	double

Format Specifier	Data Type
%c	char
%s	Strings (text), jo aap aage ke chapter mein sikhenge

C Character Data Types

The char Type

char data type ka use ek single character ko store karne ke liye hota hai. Character ko single quotes ' ' mein likhna padta hai, jaise 'A' ya 'c', aur hum ise print karne ke liye %c format specifier ka use karte hain:

Example

```
char myGrade = 'A';

printf("%c", myGrade);
```

agar aap ASCII jante hain, toh aap ASCII values ka use karke kuch characters display kar sakte hain. Dhyan rahe ki yeh values quotes (''), ke bina hoti hain, kyunki yeh numbers hote hain:

Example

```
char a = 65, b = 66, c = 67;

printf("%c", a);

printf("%c", b);
```

printf("%c", c);

Tip: Sabhi ASCII values ki list hamare **ASCII Table Reference** mein mil sakti hai.

Notes on Characters

Agar aap ek se zyada characters store karna chahte hain, toh yeh sirf last character ko print karega:

Example

```
char myText = 'Hello';

printf("%c", myText);
```

Note: Char type ka use zyada characters store karne ke liye nahi karna chahiye, kyunki yeh errors generate kar sakta hai.
Agar aapko multiple characters (ya poore shabdon) ko store karna ho, toh strings ka use karein (jo aap aage ke chapter mein sikhenge):

Example

```
char myText[] = "Hello";

printf("%s", myText);
```

Numeric Types

Agar aapko koi poora number bina decimals ke store karna ho, toh int ka use karein, jaise 35 ya 1000. Aur agar aapko floating point number (decimals ke saath) store karna ho, jaise 9.99 ya 3.14515, toh float ya double ka use karein.

int

```
int myNum = 1000;

printf("%d", myNum);
```

float

```
float myNum = 5.75;

printf("%f", myNum);
```

double

```
double myNum = 19.99;

printf("%lf", myNum);
```

C Decimal Precision

Set Decimal Precision

Aapne shayad pehle hi dekha hoga ki agar aap ek floating point number print karte ho, toh output mein decimal ke baad kaafi saare digits dikhte hain:

Example

```
float myFloatNum = 3.5;

double myDoubleNum = 19.99;

printf("%f\n", myFloatNum); // Output: 3.500000

printf("%lf", myDoubleNum); // Output: 19.990000
```

Agar aap extra zeros ko hatana chahte hain (decimal precision set karna chahte hain), toh aap ek dot (.) ka use kar sakte hain, jiske baad ek number diya jaata hai jo specify karta hai ki decimal ke baad kitne digits dikhne chahiye:

Example

```
float myFloatNum = 3.5;

printf("%f\n", myFloatNum);   // Default, 6 digits after the decimal

printf("%.1f\n", myFloatNum); // Sirf 1 digit dikhaye

printf("%.2f\n", myFloatNum); // Sirf 2 digits dikhaye

printf("%.4f", myFloatNum);   // Sirf 4 digits dikhaye
```

C The Size of Operator

Get the Memory Size

Data types chapter mein humne dekha ki variable ka memory size uske type pe depend karta hai:

Data Type	Size
int	2 ya 4 bytes
float	4 bytes
double	8 bytes
char	1 byte

Memory size yeh batata hai ki ek type ko computer ke memory mein kitni space milti hai.
Actual mein kisi data type ya variable ka size (bytes mein) jaanne ke liye, aap sizeof operator ka use kar sakte hain:

Example

```
int myInt;

float myFloat;

double myDouble;

char myChar;
```

```
printf("%zu\n", sizeof(myInt));      // int ka size

printf("%zu\n", sizeof(myFloat));    // float ka size

printf("%zu\n", sizeof(myDouble));   // double ka size

printf("%zu\n", sizeof(myChar));     // char ka size
```

C Data Types Examples

Real-Life Example

Yeh ek real-life example hai jisme hum alag-alag data types ka use karke kuch items ke total cost ko calculate aur print karte hain:

Example

```
// Alag-alag data types ke variables create karte hain

int items = 50;

float cost_per_item = 9.99;

float total_cost = items * cost_per_item;

char currency = '$';



// Variables print karte hain

printf("Items ki sankhya: %d\n", items);
printf("Har item ka cost: %.2f %c\n", cost_per_item, currency);

printf("Total cost = %.2f %c\n", total_cost, currency);
```

C Type Conversion

Type Conversion

Kabhi-kabhi aapko ek data type ke value ko doosre type mein convert karna padta hai. Isse hum type conversion kehte hain.
Maan lijiye, agar aap do integers ko divide karte ho, 5 ko 2 se divide karte ho, toh aap expect karte ho ki result 2.5 hoga. Lekin kyunki hum integers ke saath kaam kar rahe hain (aur floating-point values nahi le rahe hain), neeche diya gaya example sirf 2 output karega:

Example

```
int x = 5;

int y = 2;

int sum = 5 / 2;



printf("%d", sum); // Output: 2
```

Sahi result paane ke liye, aapko yeh samajhna hoga ki type conversion kaise kaam karta hai.
C mein type conversion ke do prakar hote hain:

- **Implicit Conversion (automatically)**
- **Explicit Conversion (manually)**

Implicit Conversion

Implicit conversion automatic hota hai jab aap ek type ka value doosre type mein assign karte ho.
Maan lijiye, agar aap ek int value ko float type mein assign karte ho:

Example

```
// Automatic conversion: int to float

float myFloat = 9;



printf("%f", myFloat); // Output: 9.000000
```

Jaise aap dekh sakte hain, compiler automatically int value 9 ko float value 9.000000 mein convert kar leta hai.
Yeh thoda risky ho sakta hai, kyunki kuch situations mein aap specific values ko control nahi kar paate.
Agar reverse ho jaye - jaise float value 9.99 ko int value 9 mein convert kiya jaye - toh kya hoga?

Example

```
// Automatic conversion: float to int

int myInt = 9.99;
```

```
printf("%d", myInt); // Output: 9
```

Yaha .99 kahaan gaya? Hum chahte the ki woh value bhi hamare program mein ho! Toh isliye, is cheez ka dhyaan rakhein.
Agar aap do integers ko divide karte hain: 5 ko 2 se divide, toh aap jaante hain ki sum 2.5 hoga. Aur jaise pehle page par diya gaya tha, agar aap sum ko integer ke roop mein store karte ho, toh result sirf 2 dikhai dega. Isliye, yeh behtar hoga ki sum ko float ya double mein store kiya jaye.

Example

```
float sum = 5 / 2;

printf("%f", sum); // Output: 2.000000
```

Yeh result 2.00000 kyun hai aur 2.5 kyun nahi? Kyunki 5 aur 2 dono integers hain division mein. Is case mein, aapko manually integers ko floating-point values mein convert karna padega. (Neeche dekhiye).

Explicit Conversion

Explicit conversion manually kiya jata hai jab aap type ko parentheses () mein value ke saamne rakhte ho.
Humare example mein, hum sahi result paane ke liye manually conversion karenge:

Example

```
// Manual conversion: int to float
```

```
float sum = (float) 5 / 2;


printf("%f", sum); // Output: 2.500000
```

Aap type ko variable ke saamne bhi rakh sakte hain:

Example

```
int num1 = 5;

int num2 = 2;

float sum = (float) num1 / num2;


printf("%f", sum); // Output: 2.500000
```

Aur jab aapne "decimal precision" ke baare mein pehle chapter mein padha tha, toh aap output ko aur clean bana sakte ho extra zeros ko hata kar (agar aap chahein):

Example

```
int num1 = 5;

int num2 = 2;

float sum = (float) num1 / num2;
```

printf("%.1f", sum); // Output: 2.5

Real-Life Example

Yeh ek real-life example hai data types aur type conversion ka, jisme hum ek program banate hain jo user ke score ka percentage calculate karta hai, jo game mein maximum score ke comparison mein hota hai:

Example

```
// Set the maximum possible score in the game to 500
int maxScore = 500;

// The actual score of the user
int userScore = 423;

/* Calculate the percantage of the user's score in relation to the maximum available
score.
Convert userScore to float to make sure that the division is accurate */
float percentage = (float) userScore / maxScore * 100.0;

// Print the percentage
printf("User's percentage is %.2f", percentage);
```

1. संरचना (struct)

A **structure** is a user-defined data type in C which allows grouping different data types together. We can store multiple properties of an object in a **structure**.

Example:

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

```
struct Person
{
    char name[50];
    int age;
    float height;
};

int main() {
    struct Person p1;
    p1.age = 30;
    p1.height = 5.9;
    snprintf(p1.name, sizeof(p1.name), "John");

    printf("Name: %s, Age: %d, Height: %.2f\n", p1.name, p1.age,
p1.height);
    return 0;
}
```

Structure ka use hota hai jab hume kisi object ke multiple attributes ko ek saath store karna ho, jaise **Person** ke naam, age, aur height ko ek structure mein rakha gaya hai.

2. संघ (union)

A **union** is similar to a structure, but all members of a **union** share the same memory location. This means, at any point, only one member of the union can hold a value.

Example:

```
#include <stdio.h>

union Data {
    int i;
    float f;
    char str[20];
};

int main() {
    union Data data;
    data.i = 10;
    printf("Data.i: %d\n", data.i);
    data.f = 220.5;
    printf("Data.f: %.2f\n", data.f);

    snprintf(data.str, sizeof(data.str), "Hello World");
    printf("Data.str: %s\n", data.str);

    return 0;
}
```

Union ka use hota hai jab aapko memory ko optimize karna ho, kyunki **union** mein sabhi members ek hi memory location ko share karte hain.

3. सङ्ग्रहण (enum)

Ek enumeration (enum) aapko integral constants ko naam dene ka mauka deta hai, jisse aapka code zyada readable aur samajhne mein aasan ho jata hai. Ye ek collection hota hai related constants ka, jise aap ek naam ke saath represent kar sakte ho.

Example:

```
#include <stdio.h>

enum Day {
    Sunday = 1, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday
};

int main() {
    enum Day today = Wednesday;
    printf("Today is day number %d\n", today);
    return 0;
}
```

Enum ka use hota hai jab hume kisi specific set of values ko naam dena ho, jaise din ke naam ko numbers ke saath map karna.

4. प्रकार-वर्णन (typedef)

typedef allows creating new names for existing data types. It helps make the code more readable and manageable.

Example:

```
#include <stdio.h>

typedef unsigned int uint;
typedef unsigned long ulong;

int main() {
    uint num1 = 100;
    ulong num2 = 1000000;

    printf("num1: %u, num2: %lu\n", num1, num2);
    return 0;
}
```

typedef ka use hota hai jab aapko kisi type ka baar-baar use karna ho aur aap chahte hain ki code ko short aur clear banaya jaaye.

5. पॉइंटर और विस्तारित प्रकार (Pointers to Extended Types)

Ek pointer ka use structures ya unions ko efficiently pass karne ke liye kiya ja sakta hai, ya phir dynamically memory allocate karne ke liye bhi.

Example (Pointer to Structure):

```
#include <stdio.h>

struct Person {
    char name[50];
    int age;
};

int main() {
    struct Person p1 = {"Alice", 25};
    struct Person *pPtr = &p1;

    printf("Name: %s, Age: %d\n", pPtr->name, pPtr->age);
    return 0;
}
```

Here, **pointer** ka use karke hum structure `p1` ko refer kar rahe hain, jisse large structures ko pass karna efficient hota hai.

6. लचीलें ऐरे सदस्य (Flexible Array Members)

In **C99**, flexible array members allow defining arrays whose size is decided at runtime.

Example:

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

struct DynamicArray {
    int size;
    int array[];  // Flexible array member
};



int main() {
    int n = 5;
    struct DynamicArray *arr = malloc(sizeof(struct DynamicArray) + n *
sizeof(int));
    arr->size = n;

    for (int i = 0; i < n; i++) {
        arr->array[i] = i * 10;
    }

    for (int i = 0; i < n; i++) {
        printf("arr[%d] = %d\n", i, arr->array[i]);
    }

    free(arr);
    return 0;
}
```

Flexible Array Member ka use tab hota hai jab aapko array ka size runtime par define karna ho, jo memory allocation ko flexible banaata hai.

7. बिट-फ़ील्ड्स (Bitfields)

Bitfields allow you to allocate a specific number of bits to a variable in a structure, which is useful for memory optimization, especially in hardware-level programming.

Example:

```
#include <stdio.h>

struct Flags {
    unsigned int a : 1;
    unsigned int b : 3;
    unsigned int c : 4;
};

int main() {
    struct Flags flag = {1, 5, 10};
    printf("a = %d, b = %d, c = %d\n", flag.a, flag.b, flag.c);
    return 0;
}
```

Bitfields ka use tab hota hai jab aapko kisi structure mein specific bits allocate karni hoti hain, jaise hardware registers mein hota hai.

C Constants

Constants

Now that you have seen different types of variables in C, you should also know that sometimes you need variables that should not change.

This can be done with the `const` keyword, which makes a variable **unchangeable** and **read-only**:

Example

```
const int myNum = 15;  // myNum will always be 15
myNum = 10;  // error: assignment of read-only variable 'myNum'
```

You should always declare a variable as `const` when you have values that are unlikely to change:

Example

```
const int minutesPerHour = 60;
const int monthsInYear = 12;
```

C Operators

Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + **operator** to add together two values:

Example

int myNum = 100 + 50;

Although + operator ka use aksar do values ko add karne ke liye kiya jata hai, jaise ki upar diye gaye example mein, isse ek variable aur ek value ko ya phir ek variable aur dusre variable ko add karne ke liye bhi use kiya ja sakta hai.

Example

```
int sum1 = 100 + 50;        // 150 (100 + 50)
int sum2 = sum1 + 250;      // 400 (150 + 250)
int sum3 = sum2 + sum2;     // 800 (400 + 400)
```

C Arithmetic Operators

Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y

/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Here is an example using different arithmetic operators in one example:

Example

```
int x = 10;

int y = 3;


printf("%d\n", x + y); // 13

printf("%d\n", x - y); // 7

printf("%d\n", x * y); // 30

printf("%d\n", x / y); // 3

printf("%d\n", x % y); // 1
```

```
int z = 5;

++z;

printf("%d\n", z); // 6

--z;

printf("%d\n", z); // 5
```

Note: Jab C mein do integers ko divide kiya jata hai, toh result bhi ek integer hota hai. Jaise, 10 / 3 ka result 3 hoga. Agar aapko decimal result chahiye, toh float ya double values ka use karein, jaise 10.0 / 3.

Example

```
int a = 10;

int b = 3;

printf("%d\n", a / b);   // Integer division, result is 3


double c = 10.0;

double d = 3.0;

printf("%f\n", c / d);   // Decimal division, result is 3.333...
```

Incrementing and Decrementing

Incrementing aur decrementing programming mein kaafi common hai, khaas kar jab counters, loops, aur arrays ke saath kaam kiya jata hai (jo aap aage ke chapters mein zyada samjhenge).

++ operator ek value ko 1 se increase karta hai, jabki -- operator ek value ko 1 se decrease karta hai.

Example

```
int x = 5;

++x; // Increment x by 1

printf("%d\n", x); // 6

--x; // Decrement x by 1

printf("%d\n", x); // 5
```

Real Life Example: Counting People

Imagine karo aap ek program bana rahe ho jo count kare ki kitne log room mein enter karte hain aur kitne log room se nikalte hain. Aap ++ operator ka use kar sakte ho counter ko increase karne ke liye jab koi room mein enter kare, aur -- operator ka use kar sakte ho counter ko decrease karne ke liye jab koi room se nikalta hai:

Example

```
int peopleInRoom = 0;

// 3 people enter

peopleInRoom++;

peopleInRoom++;

peopleInRoom++;


printf("%d\n", peopleInRoom); // 3

// 1 person leaves

peopleInRoom--;

printf("%d\n", peopleInRoom); // 2
```

C Assignment Operators

Assignment Operators

Assignment operators ka use values ko variables mein assign karne ke liye kiya jata hai.
Niche diye gaye example mein, hum assignment operator (=) ka use karke value 10 ko ek variable x mein assign kar rahe hain:

Example

```
int x = 10;
```

The **addition assignment** operator (+=) adds a value to a variable:

Example

```
int x = 10;
x += 5;
```

A list of all assignment operators:

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

&=	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

Real-Life Example: Tracking Savings

Assignment operators real-life scenarios mein bhi use kiye ja sakte hain. Maan lijiye, aap += operator ka use kar sakte ho savings ko track karne ke liye jab aap apne account mein paisa add karte ho.

Example

```
int savings = 100;

savings += 50; // add 50 to savings

printf("Total savings: %d\n", savings);
```

C Comparison Operators

Comparison Operators

Comparison operators ka use do values (ya variables) ko compare karne ke liye kiya jata hai. Ye programming mein kaafi zaroori hote hain, kyunki ye humein answers find karne mein madad karte hain aur decisions lene mein help karte hain.

Comparison ka return value ya to 1 (true) hota hai, ya 0 (false), jo Boolean values kehte hain. Aap inke baare mein **Booleans** aur If..Else chapter mein zyada padhenge.

Niche diye gaye example mein, hum greater than **operator** (>) ka use kar rahe hain yeh jaanne ke liye ki kya 5, 3 se bada hai:

Example

```
int x = 5;
int y = 3;
printf("%d", x > y); // returns 1 (true) because 5 is greater than 3
```

A list of all comparison operators:

Operator	Name	Example	Description
==	Equal to	x == y	Returns 1 if the values are equal
!=	Not equal	x != y	Returns 1 if the values are not equal

>	Greater than	x > y	Returns 1 if the first value is greater than the second value
<	Less than	x < y	Returns 1 if the first value is less than the second value
>=	Greater than or equal to	x >= y	Returns 1 if the first value is greater than, or equal to, the second value
<=	Less than or equal to	x <= y	Returns 1 if the first value is less than, or equal to, the second value

Real-Life Examples

Comparison operators are often used in real-world conditions, such as checking if a person is old enough to vote:

Example

```
int age = 18;

printf("%d\n", age >= 18); // 1 (true), old enough to vote

printf("%d\n", age < 18);  // 0 (false), not old enough
```

Another common use is checking if a password is long enough:

Example

```
int passwordLength = 5;

printf("%d\n", passwordLength >= 8); // 0 (false), too short

printf("%d\n", passwordLength < 8);  // 1 (true), needs more characters
```

C Logical Operators

Logical Operators

Comparison operators ki tarah, aap logical operators ka use karke true ya false values ko test kar sakte ho.

Logical operators ka use variables ya values ke beech mein logic determine karne ke liye hota hai, jismein multiple conditions ko combine kiya jata hai.

Operator	Name	Example	Description
&&	AND	x < 5 && x < 10	Returns 1 if both statements are true
||	OR	x < 5 || x < 4	Returns 1 if one of the statements is true

!	NOT	!(x < 5 && x < 10)	Reverse the result, returns 0 if the result is 1

Real-Life Example: Login Check

Niche diya gaya example dikhata hai ki logical operators ko kaise real situation mein use kiya ja sakta hai, jaise ki login status aur access rights ko check karte waqt:

Example

```
bool isLoggedIn = true;

bool isAdmin = false;


printf("Regular user: %s\n", (isLoggedIn && !isAdmin) ? "true" : "false");

printf("Has access: %s\n", (isLoggedIn || isAdmin) ? "true" : "false");

printf("Not logged in: %s\n", (!isLoggedIn) ? "true" : "false");
```

Result:

```
Regular user: true
Has access: true
Not logged in: false
```

C Operator Precedence

Operator Precedence

Jab kisi calculation mein ek se zyada operator hote hain, C order of operations rules follow karta hai yeh decide karne ke liye ki pehle kis part ko calculate kiya jaye.

Example ke liye, multiplication addition se pehle hota hai:

Example

```
int result1 = 2 + 3 * 4;      // 2 + 12 = 14
int result2 = (2 + 3) * 4;    // 5 * 4 = 20


printf("%d\n", result1);
printf("%d\n", result2);
```

Why Does This Happen?

In `2 + 3 * 4`, the multiplication is done first, so the answer is `14`.

If you want the addition to happen first, you must use parentheses: `(2 + 3) * 4`, which gives `20`.

Tip: Always use parentheses `( )` if you want to make sure the calculation is done in the order you expect. It also makes your code easier to read.

Order of Operations

Here are some common operators in C, from highest to lowest priority:

- `()` - Parentheses
- `*`, `/`, `%` - Multiplication, Division, Modulus
- `+`, `-` - Addition, Subtraction
- `>`, `<`, `>=`, `<=` - Comparison
- `==`, `!=` - Equality
- `&&` - Logical AND
- `||` - Logical OR
- `=` - Assignment

Another Example

Subtraction and addition are done from left to right, unless you add parentheses:

Example

```
int result1 = 10 - 2 + 5;    // (10 - 2) + 5 = 13

int result2 = 10 - (2 + 5);  // 10 - 7 = 3


printf("%d\n", result1);

printf("%d\n", result2);
```

C Booleans

Booleans

Programming mein kaafi baar aapko ek aise data type ki zaroorat padti hai jo sirf do values mein se ek hi value rakh sake, jaise:

- YES / NO
- ON / OFF
- TRUE / FALSE

Iske liye, C mein bool data type hota hai, jo booleans ke naam se jaana jata hai.
Booleans do values mein se ek ko represent karte hain: true ya false.

Boolean Variables

In C, the `bool` type is not a built-in data type, like `int` or `char`.

It was introduced in C99, and you must **import** the following header file to use it:

```
#include <stdbool.h>
```

A boolean variable is declared with the `bool` keyword and can take the values `true` or `false`:

```
bool isProgrammingFun = true;
bool isFishTasty = false;
```

Before trying to print the boolean variables, you should know that boolean values are returned as **integers**:

- `1` (or any other number that is not 0) represents `true`
- `0` represents `false`

Therefore, you must use the %d format specifier to print a **boolean** value:

Example

```
// Create boolean variables
bool isProgrammingFun = true;
bool isFishTasty = false;

// Return boolean values
printf("%d", isProgrammingFun);   // Returns 1 (true)
printf("%d", isFishTasty);        // Returns 0 (false)
```

However, it is more common to return a **boolean** value by **comparing** values and variables.

Comparing Values and Variables

Comparing values are useful in programming, because it helps us to find answers and make decisions.

For example, you can use a comparison operator, such as the **greater than** (>) operator, to compare two values:

Example

```
printf("%d", 10 > 9);  // Returns 1 (true) because 10 is greater than 9
```

From the example above, you can see that the return value is a boolean value (1).

You can also compare two variables:

Example

```
int x = 10;
int y = 9;
printf("%d", x > y);
```

In the example below, we use the **equal to** (==) operator to compare different values:

Example

```
printf("%d", 10 == 10); // Returns 1 (true), because 10 is equal to 10
printf("%d", 10 == 15); // Returns 0 (false), because 10 is not equal to 15
printf("%d", 5 == 55);  // Returns 0 (false) because 5 is not equal to 55
```

You are not limited to only compare numbers. You can also compare boolean variables, or even special structures, like arrays (which you will learn more about in a later chapter):

Example

```
bool isHamburgerTasty = true;
bool isPizzaTasty = true;

// Find out if both hamburger and pizza is tasty
printf("%d", isHamburgerTasty == isPizzaTasty);
```

C Boolean Examples

Real Life Example

Let's think of a "real life example" where we need to find out if a person is old enough to vote.

In the example below, we use the `>=` comparison operator to find out if the age (`25`) is **greater than** OR **equal to** the voting age limit, which is set to `18`:

Example

```
int myAge = 25;
int votingAge = 18;

printf("%d", myAge >= votingAge); // Returns 1 (true), meaning 25 year olds are
allowed to vote!
```

Cool, right? An even better approach (since we are on a roll now), would be to wrap the code above in an `if...else` statement, so we can perform different actions depending on the result:

Example

Output "Old enough to vote!" if `myAge` is **greater than or equal to** `18`. Otherwise output "Not old enough to vote.":

```
int myAge = 25;
int votingAge = 18;
if (myAge >= votingAge) {
  printf("Old enough to vote!");
} else {
  printf("Not old enough to vote.");
}
```

Booleans are the basis for all comparisons and conditions.

You will learn more about conditions (`if...else`) in the next chapter.

C If ... Else

Conditions and If Statements

You already know that C supports familiar comparison conditions from mathematics, such as:

- Less than: `a < b`
- Less than or equal to: `a <= b`

- Greater than: `a > b`
- Greater than or equal to: `a >= b`
- Equal to `a == b`
- Not Equal to: `a != b`

You can use these conditions to perform different actions for different decisions.

C has the following conditional statements:

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

The if Statement

Use the `if` statement to specify a block of code to be executed if a condition is `true`.

Syntax

```
if (condition) {
  // block of code to be executed if the condition is true
}
```

Note that `if` is in lowercase letters. Uppercase letters (If or IF) will generate an error.

In the example below, we test two values to find out if 20 is greater than 18. If the condition is `true`, print some text:

Example

```
if (20 > 18) {
  printf("20 is greater than 18");
}
```

We can also test variables:

Example

```
int x = 20;
int y = 18;
if (x > y) {
  printf("x is greater than y");
}
```

C Else

The else Statement

Use the `else` statement to specify a block of code to be executed if the condition is `false`.

Syntax

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

Example

```
int time = 20;
if (time < 18) {
  printf("Good day.");
} else {
  printf("Good evening.");
}
// Outputs "Good evening."
```

C Else If

The else if Statement

Use the `else if` statement to specify a new condition if the first condition is `false`.

Syntax

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

Example

```
int time = 22;
if (time < 10) {
  printf("Good morning.");
} else if (time < 20) {
  printf("Good day.");
} else {
  printf("Good evening.");
}
// Outputs "Good evening."
```

Example explained

In the example above, time (22) is greater than 10, so the **first condition** is `false`. The next condition, in the `else if` statement, is also `false`, so we move on to the `else` condition since **condition1** and **condition2** is both `false` - and print to the screen "Good evening".

However, if the time was 14, our program would print "Good day."

C Short Hand If Else

Short Hand If...Else (Ternary Operator)

There is also a short-hand if else, which is known as the **ternary operator** because it consists of three operands. It can be used to replace multiple lines of code with a single line. It is often used to replace simple if else statements:

Syntax

```
variable = (condition) ? expressionTrue : expressionFalse;
```

Instead of writing:

Example

```
int time = 20;
if (time < 18) {
  printf("Good day.");
} else {
  printf("Good evening.");
}
```

You can simply write:

Example

```
int time = 20;
(time < 18) ? printf("Good day.") : printf("Good evening.");
```

C Nested If

Nested If

You can also place an `if` statement inside another `if`. This is called a **nested if** statement.

A nested `if` lets you check for a condition only if another condition is already `true`.

Syntax

```
if (condition1) {
  // code to run if condition1 is true
  if (condition2) {
    // code to run if both condition1 and condition2 are true
  }
}
```

Example

In this example, we first check if x is greater than 10. If it is, we then check if y is greater than 20:

Example

```
int x = 15;

int y = 25;

if (x > 10) {

  printf("x is greater than 10\n");

  // Nested if

if (y > 20) {

    printf("y is also greater than 20\n");

  }

}
```

Result:

[illegible]

Real-Life Example

Nested `if` statements are useful when you need to test multiple conditions that depend on each other. For example, checking if a person is old enough to vote, and if they are a citizen:

Example

```
int age = 20;
bool isCitizen = true;

if (age >= 18) {
  printf("Old enough to vote.\n");

  if (isCitizen) {
    printf("And you are a citizen, so you can vote!\n");
  } else {
    printf("But you must be a citizen to vote.\n");
  }
} else {
  printf("Not old enough to vote.\n");
}
```

Result:

C Logical Operators in Conditions

Logical Operators in Conditions

You can combine or reverse conditions using logical operators. These work together with `if`, `else`, and `else if` to build more complex decisions.

- `&&` (AND) - all conditions must be true
- `||` (OR) - at least one condition must be true
- `!` (NOT) - reverses a condition (`true` → `false`, `false` → `true`)

AND (&&)

Use AND (`&&`) when **both** conditions must be true:

Example

Test if `a` is greater than `b`, **and** if `c` is greater than `a`:

```
int a = 200;

int b = 33;

int c = 500;


if (a > b && c > a) {

  printf("Both conditions are true\n");

}
```

OR (||)

Use OR (||) when **at least one** of the conditions can be true:

Example

Test if `a` is greater than `b`, **or** if `a` is greater than `c`:

```
int a = 200;

int b = 33;

int c = 500;


if (a > b || a > c) {

  printf("At least one condition is true\n");

}
```

NOT (!)

The NOT operator (`!`) **reverses** a condition:

- If a condition is `true`, `!` makes it `false`.
- If a condition is `false`, `!` makes it `true`.

This is useful when you want to check that something *is not* the case:

Example

Test if `a` is **not** greater than `b`:

```
int a = 33;

int b = 200;


if (!(a > b)) {

  printf("a is NOT greater than b\n");

}
```

Real-Life Example

In real programs, logical operators are often used for access control. For example, to get access to a system, there are specific requirements:

You must be logged in, and then you either need to be an admin, or have a high security clearance (level 1 or 2):

Example

```
bool isLoggedIn = true;

bool isAdmin = false;

int securityLevel = 3; // 1 = highest


if (isLoggedIn && (isAdmin || securityLevel <= 2)) {

  printf("Access granted\n");

} else {
  printf("Access denied\n");
}


// Try changing securityLevel and isAdmin to test different
outcomes:

// securityLevel 1 = Access granted

// securityLevel 2 = Access granted

// securityLevel 3 = Access denied

// securityLevel 4 = Access denied

// If isAdmin = true, access is granted.
```

C If ... Else Examples

Real-Life Examples

This example shows how you can use `if..else` to "open a door" if the user enters the correct code:

Example

```
int doorCode = 1337;

if (doorCode == 1337) {
  printf("Correct code.\nThe door is now open.");
} else {
  printf("Wrong code.\nThe door remains closed.");
}
```

This example shows how you can use `if..else` to find out if a number is positive or negative:

Example

```
int myNum = 10; // Is this a positive or negative number?

if (myNum > 0) {
  printf("The value is a positive number.");
} else if (myNum < 0) {
  printf("The value is a negative number.");
} else {
  printf("The value is 0.");
}
```

Find out if a person is old enough to vote:

Example

```
int myAge = 25;
int votingAge = 18;

if (myAge >= votingAge) {
  printf("Old enough to vote!");
} else {
  printf("Not old enough to vote.");
}
```

Find out if a person is old enough to vote, **and** if they are a citizen (using nested if statements):

Example

```
int age = 20;

bool isCitizen = true;


if (age >= 18) {

  printf("Old enough to vote.\n");


  if (isCitizen) {

    printf("And you are a citizen, so you can vote!\n");

  } else {

    printf("But you must be a citizen to vote.\n");

  }

} else {

  printf("Not old enough to vote.\n");

}
```

Find out if a number is even or odd:

Example

```
int myNum = 5;

if (myNum % 2 == 0) {
  printf("%d is even.\n", myNum);
} else {
  printf("%d is odd.\n", myNum);
}
```

Check temperature (Celsius):

Example

```
int temperature = 30;

if (temperature < 0) {
  printf("It's freezing!\n");
} else if (temperature < 20) {
  printf("It's cool.\n");
} else {
  printf("It's warm.\n");
}
```

System access control example - You must be logged in, and then you either need to be an admin, or have a high security clearance (level 1 or 2) to get access:

Example

```
bool isLoggedIn = true;

bool isAdmin = false;
```

```
int securityLevel = 3; // 1 = highest


if (isLoggedIn && (isAdmin || securityLevel <= 2)) {

  printf("Access granted\n");

} else {

  printf("Access denied\n");

}


// Try changing securityLevel and isAdmin to test different
outcomes:

// securityLevel 1 = Access granted

// securityLevel 2 = Access granted

// securityLevel 3 = Access denied

// securityLevel 4 = Access denied

// If isAdmin = true, access is granted.
```

C Switch

Switch Statement

Instead of writing **many** `if..else` statements, you can use the `switch` statement.

The `switch` statement selects one of many code blocks to be executed:

Syntax

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

This is how it works:

- The `switch` expression is evaluated once
- The value of the expression is compared with the values of each **case**
- If there is a match, the associated block of code is executed
- The `break` statement breaks out of the switch block and stops the execution
- The `default` statement is optional, and specifies some code to run if there is no case match

The example below uses the weekday number to calculate the weekday name:

Example

```
int day = 4;

switch (day) {
  case 1:
    printf("Monday");
    break;
  case 2:
    printf("Tuesday");
    break;
  case 3:
    printf("Wednesday");
    break;
  case 4:
    printf("Thursday");
    break;
  case 5:
    printf("Friday");
    break;
  case 6:
    printf("Saturday");
    break;
  case 7:
    printf("Sunday");
    break;
}

// Outputs "Thursday" (day 4)
```

The break Keyword

When C reaches a `break` keyword, it breaks out of the switch block.

This will stop the execution of more code and case testing inside the block.

When a match is found, and the job is done, it's time for a break. There is no need for more testing.

A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.

The default Keyword

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

Example

```
int day = 4;

switch (day) {
  case 6:
    printf("Today is Saturday");
    break;
  case 7:
    printf("Today is Sunday");
    break;
  default:
    printf("Looking forward to the Weekend");
}

// Outputs "Looking forward to the Weekend"
```

🔁 **`while`** Loop in C Language

---

1. **`while`** loop kya hota hai?

`while` loop ek **entry-controlled loop** hai. Iska matlab hai ki condition **pehle check hoti hai**, agar true hoti hai to body of loop execute hoti hai. Agar condition pehle hi false ho, to loop **ek baar bhi execute nahi hota**.

---

2. Syntax of `while` loop

```
while(condition) {
    // loop body
    // statements to execute
}
```

👉 Yahaan par:

- condition: ye ek expression hota hai jo true ya false hota hai.
- Jab tak condition true hai, tab tak loop chalta rahega.
- Jaise hi condition false hoti hai, loop ruk jaata hai.

---

1. Flowchart (kaise kaam karta hai)

```
+-------------------+
|   Condition?      |
+--------+----------+
         |
  Yes    |      No
 +-------v-------+
 |  Loop Body    |
 +---------------+
      |
Go back to check condition
```

---

4. Example of `while` loop

```
#include <stdio.h>

int main() {
    int i = 1;

    while(i <= 5) {
        printf("%d\n", i);
        i++;
    }

    return 0;
}
```

Output:

```
1
2
3
4
5
```

Yahaan par:

- `i` ki value 1 se start ho rahi hai.
- Jab tak `i <= 5` true hai, tab tak loop chalta hai.
- Har baar `i++` se `i` badhta hai.
- Jab `i` 6 ho jaata hai, `i <= 5` false ho jaata hai, aur loop ruk jaata hai.

5. Infinite while loop

Agar condition kabhi false hi nahi hoti, to loop **kabhi nahi rukta** — ise kehte hain **infinite loop**.

```
while(1) {
    // Ye loop kabhi nahi rukega
    printf("Hello\n");
}
```

6. Common mistakes

1. **Loop control variable ko update karna bhool jaana:**
2. `int i = 1;`
3. `while(i <= 5) {`
4. `    printf("%d\n", i);`
5. `    // i++ bhool gaye to infinite loop`
6. `}`
7. **Wrong condition likhna:**
8. `while(i = 5) // Galat: '=' assignment hai, '==' comparison hona chahiye`

7. Real-life Use Cases

- Repeated user input lena
- Menu-driven programs banana
- Kisi file ka end tak data read karna
- Kisi task ko condition ke complete hone tak repeat karna

8. Difference: `while` vs `do-while`

Feature	`while` loop	`do-while` loop
Condition check	Pehle (entry control)	Baad mein (exit control)
Minimum execution	0 times	At least 1 time

Example:

```
int i = 10;
while(i < 5) {
    printf("Hello"); // Ye kabhi nahi chalega
}

do {
    printf("Hello"); // Ye ek baar chalega
} while(i < 5);
```

Summary (Short Points)

- `while` ek entry-controlled loop hai.
- Condition pehle check hoti hai.
- Agar condition true hai, loop body execute hoti hai.
- Condition false hone par loop rukta hai.
- Infinite loop se bachne ke liye condition aur update dhyan se likho.

C Do/While Loop

The Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

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

The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

Example

```
int i = 0;

do {
  printf("%d\n", i);
  i++;
}
while (i < 5);
```

Do not forget to increase the variable used in the condition, otherwise the loop will never end!

Condition is False from the Start

In the example above, the condition `i < 5` was **true** at the beginning, so the loop executed multiple times. But what if the condition is **false** right from the start?

In the example below, the variable `i` starts at `10`, so the condition `i < 5` is false immediately - yet the `do/while` loop still runs once:

Example

Even if the condition is false from the start, the code block will still execute one time:

```
int i = 10;

do {
  printf("i is %d\n", i);
  i++;
} while (i < 5);
```

Summary

The `do/while` loop always runs at least once, even if the condition is already false. This is different from a regular `while` loop, which would skip the loop entirely if the condition is false at the start.

This behavior makes `do/while` useful when you want to ensure something happens at least once, like showing a message or asking for user input.

Practical Example: User Input

This example keeps asking the user to enter a positive number. The loop stops when the user enters 0 or a negative number:

Example

```
int number;

do {
  printf("Enter a positive number: ");
  scanf("%d", &number);
} while (number > 0);
```

C While Loop Examples

Real-Life Examples

To demonstrate a practical example of the **while loop**, we have created a simple "countdown" program:

Example

```
int countdown = 3;

while (countdown > 0) {
  printf("%d\n", countdown);
  countdown--;
}

printf("Happy New Year!!\n");
```

In this example, we create a program that only print even numbers between 0 and 10 (inclusive):

Example

```
int i = 0;

while (i <= 10) {
  printf("%d\n", i);
  i += 2;
}
```

In this example we use a while loop to reverse some numbers:

Example

```
// A variable with some specific numbers
int numbers = 12345;

// A variable to store the reversed number
int revNumbers = 0;

// Reverse and reorder the numbers
while (numbers) {
  // Get the last number of 'numbers' and add it to 'revNumber'
  revNumbers = revNumbers * 10 + numbers % 10;
  // Remove the last number of 'numbers'
  numbers /= 10;
}
```

To demonstrate a practical example of the **while loop** combined with an **if else statement**, let's say we play a game of Yatzy:

Example

Print "Yatzy!" If the dice number is 6:

```
int dice = 1;

while (dice <= 6) {
  if (dice < 6) {
    printf("No Yatzy\n");
  } else {
    printf("Yatzy!\n");
  }
  dice = dice + 1;
}
```

If the loop passes the values ranging from 1 to 5, it prints "No Yatzy". Whenever it passes the value 6, it prints "Yatzy!".

C For Loop

For Loop

C programming mein **`for`** loop ka use repetition ya iteration ke liye kiya jata hai. Jab hum kisi task ko baar-baar perform karna chahte hain, tab hum **`for`** loop ka use karte hain.

`for` loop ka syntax:

```
for(initialization; condition; update) {
    // loop body
}
```

1. Initialization:

Yahaan hum variable ko set karte hain jo loop ke start hone se pehle ek baar execute hota hai.

2. Condition:

Yahaan ek condition di jaati hai jo har iteration ke baad check hoti hai. Agar condition `true` hoti hai, to loop continue hota hai; agar `false` hoti hai, to loop terminate ho jata hai.

3. Update:

Yeh part loop ki har iteration ke baad execute hota hai, aur usually variable ko increment ya decrement karta hai.

Example:

```
#include <stdio.h>

int main() {
    // Example: Print numbers from 1 to 5
    for(int i = 1; i <= 5; i++) {
        printf("%d\n", i);  // Print current value of i
    }
    return 0;
}
```

Explanation:

- **Initialization**: `int i = 1;` — Loop ke start hone pe `i` ko 1 se set kar rahe hain.

- **Condition**: `i <= 5;` — Jab tak `i` 5 se chhota ya barabar hai, loop chalega.
- **Update**: `i++` — Har iteration ke baad `i` ka value 1 se increase hoga.

Output:

```
1
2
3
4
5
```

Important Points:

1. **Initialization**: Yeh loop ke pehle execute hota hai aur sirf ek baar hota hai.
2. **Condition**: Har iteration ke baad condition check hoti hai. Agar condition `false` ho jaaye, to loop stop ho jata hai.
3. **Update**: Yeh part har loop iteration ke baad execute hota hai aur variable ko update karta hai.

Nested `for` Loop:

Agar aapko do ya zyada loops ko ek saath run karna ho, toh aap nested loops ka use kar sakte hain.

Example:

```
#include <stdio.h>

int main() {
    for(int i = 1; i <= 3; i++) {
        for(int j = 1; j <= 3; j++) {
            printf("i = %d, j = %d\n", i, j);
        }
    }
    return 0;
}
```

Output:

```
i = 1, j = 1
i = 1, j = 2
i = 1, j = 3
i = 2, j = 1
i = 2, j = 2
i = 2, j = 3
i = 3, j = 1
i = 3, j = 2
i = 3, j = 3
```

Yahaan pe, pehla loop `i` ko 1 se lekar 3 tak iterate karega, aur har iteration mein dusra loop `j` ko 1 se lekar 3 tak iterate karega.

Simple `for` Loop Example:

```
#include <stdio.h>

int main() {
    // Simple loop to print numbers from 1 to 10
    for(int i = 1; i <= 10; i++) {
        printf("%d\n", i);  // Print current value of i
    }

    return 0;
}
```

Explanation:

- **Initialization**: `int i = 1;` — `i` ko 1 se initialize kiya gaya hai.
- **Condition**: `i <= 10;` — Loop tab tak chalega jab tak `i` 10 ya usse chhota hai.
- **Update**: `i++` — Har iteration ke baad `i` ki value 1 se badhegi.

Output:

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

Process:

- Pehli baar jab loop start hota hai, `i` 1 hota hai.
- Condition `i <= 10` true hoti hai, to `printf("%d\n", i);` execute hota hai aur `i` ko print karta hai.
- Uske baad, `i++` ke through `i` ka value 1 se badhkar 2 ho jata hai, aur yeh process tab tak chalti hai jab tak `i` 10 tak nahi pahuchta.

Example: Multiplication Table (1 to 10)

```
#include <stdio.h>

int main() {
    int num;

    // User se number input lena
    printf("Enter a number to print its table: ");
    scanf("%d", &num);

    // Multiplication table print karna
    for(int i = 1; i <= 10; i++) {
        printf("%d x %d = %d\n", num, i, num * i);
    }

    return 0;
}
```

Explanation:

1. **Input**: User se ek number input lene ke liye `scanf` function use kiya gaya hai.
2. **Loop**: `for` loop ko 1 se 10 tak iterate karaya gaya hai. Har iteration mein hum `num * i` ka result print karte hain.

Example Input and Output:

Input:

```
Enter a number to print its table: 5
```

Output:

```
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

Process:

- User ne number 5 diya hai, to loop 1 se lekar 10 tak chalega aur 5 ka table print karega.

Example 1: Right-Angled Triangle (Star Pattern)

Is pattern mein stars (*) ek right-angled triangle ki tarah print hote hain.

Code:

```
#include <stdio.h>

int main() {
    int n = 5;   // Number of rows

    // Star pattern print karna
    for(int i = 1; i <= n; i++) {   // Outer loop for rows
        for(int j = 1; j <= i; j++) {   // Inner loop for columns
            printf("* ");
        }
        printf("\n");   // Next row ke liye new line
    }

    return 0;
}
```

Output:

```
*
* *
* * *
* * * *
* * * * *
```

Explanation:

- **Outer Loop (`i`)**: Yeh loop rows ke liye hai, jo 1 se lekar `n` tak chal raha hai.
- **Inner Loop (`j`)**: Yeh loop har row mein stars ko print karta hai. Har row mein number of stars barhta jata hai.
- **`printf("\n");`**: Yeh har row ke baad new line print karta hai.

Example 2: Inverted Right-Angled Triangle (Star Pattern)

Is pattern mein stars ek inverted right-angled triangle ki tarah print hote hain.

Code:

```
#include <stdio.h>

int main() {
    int n = 5;   // Number of rows

    // Inverted star pattern print karna
    for(int i = n; i >= 1; i--) {  // Outer loop for rows (reverse)
        for(int j = 1; j <= i; j++) {  // Inner loop for columns
            printf("* ");
        }
        printf("\n");  // Next row ke liye new line
    }
    return 0;
}
```

Output:

```
* * * * *
* * * *
* * *
* *
*
```

Explanation:

- **Outer Loop (`i`)**: Yeh loop rows ko reverse order mein print karta hai, starting from `n` and going down to `1`.
- **Inner Loop (`j`)**: Har row mein stars ko print karta hai, jo row number ke hisaab se decrease hote hain.
- **`printf("\n");`**: Har row ke baad new line print hota hai.

Example 3: Pyramid Pattern (Star Pattern)

Yeh pattern ek pyramid shape jaisa dikhta hai, jisme har row mein stars ki count barhti jati hai aur center mein alignment hota hai.

Code:

```
#include <stdio.h>

int main() {
    int n = 5;  // Number of rows

    // Pyramid pattern print karna
    for(int i = 1; i <= n; i++) {
        // Spaces print karna (for center alignment)
        for(int j = i; j < n; j++) {
            printf(" ");  // Print space
        }

        // Stars print karna
        for(int j = 1; j <= (2*i - 1); j++) {
            printf("*");  // Print star
        }

        // Next row ke liye new line
        printf("\n");
    }

    return 0;
}
```

Output:

```
    *
   ***
  *****
 *******
*********
```

Explanation:

- **Outer Loop (`i`)**: Yeh loop rows ko handle karta hai (1 se 5 tak).
- **First Inner Loop (`j`)**: Spaces ko print karta hai taaki stars center mein aligned ho sakein.
- **Second Inner Loop (`j`)**: Stars ko print karta hai. Har row mein stars ka count `2*i - 1` hota hai.
- **`printf("\n");`**: Har row ke baad new line print hoti hai.

Example 4: Diamond Pattern (Star Pattern)

Yeh pattern ek diamond shape mein stars ko print karta hai.

Code:

```c
#include <stdio.h>

int main() {
    int n = 5;   // Number of rows

    // Upper half of the diamond
    for(int i = 1; i <= n; i++) {
        // Spaces for upper half
        for(int j = i; j < n; j++) {
            printf(" ");
        }

        // Stars for upper half
        for(int j = 1; j <= (2*i - 1); j++) {
            printf("*");
        }

        printf("\n");   // Next row ke liye new line
    }

    // Lower half of the diamond
    for(int i = n-1; i >= 1; i--) {
        // Spaces for lower half
        for(int j = n; j > i; j--) {
            printf(" ");
        }

        // Stars for lower half
        for(int j = 1; j <= (2*i - 1); j++) {
            printf("*");
        }

        printf("\n");   // Next row ke liye new line
    }

    return 0;
}
```

Output:

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

Explanation:

- **Upper Half**: Yeh part pyramid pattern jaisa hota hai, jisme spaces aur stars ka arrangement hota hai.
- **Lower Half**: Yeh part inverted pyramid ki tarah hota hai, jisme stars decrease hote hain aur spaces increase hoti hain.
- **Spaces aur Stars ka combination**: Har row mein spaces aur stars ka combination is tarah se hota hai ki diamond shape form hota hai.

C Break and Continue

Break

C language mein `break` aur `continue` statement ka use control flow ko manage karne ke liye hota hai. Chalo, inhe ek example ke saath samajhte hain:

1. `break` Statement:

`break` ka use kisi loop ko **forcefully** break karne ke liye kiya jata hai. Jab `break` statement execute hota hai, tab loop turant terminate ho jata hai, chahe loop ka condition true ho ya false.

Example:

```
#include <stdio.h>

int main() {
    for (int i = 0; i < 10; i++) {
        if (i == 5) {
            break;  // Loop yahan break ho jayega jab i 5 ho
jayega
        }
        printf("%d ", i);
    }
    return 0;
}
```

Output:

```
0 1 2 3 4
```

Yahan, jaise hi `i` ki value 5 hoti hai, `break` statement execute hota hai aur loop terminate ho jata hai.

Continue

2. `continue` Statement:

`continue` ka use kisi loop ko **skip** karne ke liye hota hai. Matlab, jab `continue` statement execute hota hai, to loop ka current iteration **skip** ho jata hai aur loop apna next iteration start kar leta hai.

Example:

```
#include <stdio.h>

int main() {
    for (int i = 0; i < 10; i++) {
        if (i == 5) {
            continue;  // Jab i 5 ho, to iteration skip ho
jayega
        }
        printf("%d ", i);
    }
    return 0;
}
```

Output:

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

Yahan, jab `i` ki value 5 hoti hai, to `continue` statement execute hota hai, aur 5 ko print nahi hota. Baaki iterations normal chalti hain.

Combining Break and Continue

You can also combine `break` and `continue`.

This example skips printing 2 and stops the loop at 4:

Example

```
int i;

for (i = 0; i < 6; i++) {
  if (i == 2) {
    continue;
  }
  if (i == 4) {
    break;
  }
  printf("%d\n", i);
}
```

Break and Continue in While Loop

You can also use break and continue in while loops:

Break Example

```
int i = 0;
```

```
while (i < 10) {
  if (i == 4) {
    break;
  }
  printf("%d\n", i);
  i++;
}
```

Continue Example

```
int i = 0;

while (i < 10) {
  if (i == 4) {
    i++;
    continue;
  }
  printf("%d\n", i);
  i++;
}
```

Real-Life Example

Imagine processing a list of numbers where you want to skip negative values, but stop completely if you find a zero:

Example

```
int myNumbers[] = {3, -1, 7, 0, 9};

int length = sizeof(myNumbers) / sizeof(myNumbers[0]);

int i;


for (i = 0; i < length; i++) {

  if (myNumbers[i] < 0) {

    continue; // skip negative numbers

  }

  if (myNumbers[i] == 0) {

    break; // stop loop when zero is found

  }

  printf("%d\n", myNumbers[i]);

}
```

Summary:

- `break`: Loop ko turant terminate karne ke liye use hota hai.
- `continue`: Current iteration ko skip karne ke liye use hota hai, aur loop ka next iteration continue hota hai.

C Arrays

Arrays

C language mein **array** ek collection hota hai same type ke elements ka, jo contiguous memory locations mein store hote hain. Array ko aap ek fixed size ka container samajh sakte hain jisme aap multiple values ko store kar sakte hain.

Array Declaration:

Array ko declare karte waqt uska type, naam, aur size define karna padta hai. Size bataata hai ki array mein kitne elements store kar sakte hain.

Syntax:

```
datatype array_name[size];
```

- **`datatype`:** Array ke elements ka type, jaise `int`, `float`, `char`, etc.
- **`array_name`:** Array ka naam.
- `size`: Array ke elements ki number of items.

Example:

```
int numbers[5];  // Array of 5 integers
char name[20];   // Array of 20 characters
```

Array Initialization:

Aap array ko values assign karte waqt bhi initialize kar sakte ho.

Example:

```
int numbers[5] = {1, 2, 3, 4, 5};  // Array of 5 integers
initialized
char name[6] = "Hello";  // Array of 6 characters, including
null terminator
```

Agar array ko initialize nahi kiya, to array ke elements undefined rahte hain (garbage values).

Accessing Array Elements:

Array ke elements ko index ke through access kiya jata hai. Index **0-based** hota hai, yani first element ka index `0` hota hai.

Example:

```
#include <stdio.h>

int main() {
    int numbers[5] = {10, 20, 30, 40, 50};

    printf("First element: %d\n", numbers[0]);  // 10
    printf("Third element: %d\n", numbers[2]); // 30

    return 0;
}
```

Output:

```
First element: 10
Third element: 30
```

Array Example with Loop:

Arrays ko loop ke saath bhi use kiya ja sakta hai. Yahan ek example diya gaya hai jisme `for` loop ke through array ke saare elements ko print kiya gaya hai.

Example:

```
#include <stdio.h>

int main() {
    int numbers[5] = {1, 2, 3, 4, 5};

    // Loop through array and print each element
    for (int i = 0; i < 5; i++) {
        printf("%d ", numbers[i]);
    }

    return 0;
}
```

Output:

```
1 2 3 4 5
```

Example :

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

```
int main() {
    int myNumbers[] = {10, 20, 30, 40, 50};
    int length = sizeof(myNumbers) / sizeof(myNumbers[0]);
// elements ka count nikalna

    // Loop ke through array ke elements print kar rahe hain
    for(int i = 0; i < length; i++) {
        printf("%d\n", myNumbers[i]);
    }

    return 0;
}
```

Explanation (Hinglish mein):

- `int myNumbers[] = {10, 20, 30, 40, 50};`
  → Ye ek integer array hai jisme 5 numbers store kiye gaye hain.
- `sizeof(myNumbers) / sizeof(myNumbers[0]);`
  → Ye formula array ke total elements ka count batata hai.
- `for(int i = 0; i < length; i++)`
  → Ye loop 0 se lekar (length - 1) tak chalega, matlab array ke har element tak.
- `printf("%d\n", myNumbers[i]);`
  → Ye line har element ko ek-ek karke print karegi.

Output:

```
10
20
30
40
50
```

C Array Size

Get Array Size

Array Size Kaise Pata Karein

Array ka size pata karne ke liye, aap `sizeof` operator ka use kar sakte hain:

Example

```
int myNumbers[] = {10, 25, 50, 75, 100};
printf("%zu", sizeof(myNumbers));  // 20 print hoga
```

Note:

Ye result 20 kyun dikh raha hai, jabki array mein 5 elements hain?

- Iska reason ye hai ki `sizeof` operator type ka size bytes mein return karta hai.

Aapne Data Types chapter mein sikha tha ki `int` type ka size usually 4 bytes hota hai. Isliye example ke hisaab se, 4 x 5 (4 bytes x 5 elements) = 20 bytes.

Jab aap bade programs par kaam karte hain jahan memory management zaroori ho, tab array ka memory size jaanana kaafi helpful hota hai.

Lekin agar aapko sirf ye jaana ho ki array mein kitne elements hain?

Elements Ki Sankhya Pata Karna

Agar aap array ke elements ka count chahte hain, to aap yeh formula use kar sakte hain, jo total size ko ek element ke size se divide karta hai:

Example

```
int myNumbers[] = {10, 25, 50, 75, 100};
int length = sizeof(myNumbers) / sizeof(myNumbers[0]);

printf("%d", length);  // Prints 5
```

formula arrays ke kisi bhi type aur size ke liye kaam karta hai:

Example

```
double myValues[] = {1.1, 2.2, 3.3};
int length = sizeof(myValues) / sizeof(myValues[0]);

printf("%d", length);  // 3 print hoga
```

Multi-Dimensional Arrays:

C mein arrays ko multi-dimensional bhi banaya ja sakta hai, jaise 2D array.

Example:

```
#include <stdio.h>

int main() {
    int matrix[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };

    // Accessing elements of 2D array
    printf("Element at [0][0]: %d\n", matrix[0][0]);  // 1
    printf("Element at [1][2]: %d\n", matrix[1][2]);  // 6

    return 0;
}
```

Output:

```
Element at [0][0]: 1
Element at [1][2]: 6
```

Array Key Points:

- **Indexing:** Array ke elements ko index ke through access karte hain, jo 0 se start hota hai.
- **Fixed Size:** Ek array ka size fixed hota hai jab aap usko declare karte ho.
- **Homogeneous:** Array mein sabhi elements same type ke hote hain (e.g., `int`, `float`, etc.).
- **Memory Allocation:** Array ek contiguous block mein memory allocate karta hai.

Array Limitation:

1. **Size Fixed:** Ek baar array declare karne ke baad, uska size change nahi kar sakte.
2. **Memory Consumption:** Agar array ka size bahut bada hai, to memory ka wastage ho sakta hai.

C STRINGS

STRINGS

STRINGS KA USE TEXT YA CHARACTERS KO STORE KARNE KE LIYE HOTA HAI.

For example, **"Hello World"** ek string hai jo characters ka combination hai.

C mein, kai dusre programming languages ke comparison mein, koi `String` **type** nahi hota jisse aap easily string variables create kar sakein. Iske bajaye, aapko `char` type use karna padta hai aur characters ka ek array banana padta hai string banane ke liye:

```
char greetings[] = "Hello World!";
```

Note: Aapko double quotes (`""`) ka use karna padta hai string ko define karte waqt.

Output the String

String ko output karne ke liye, aap `printf()` function ka use karte hain aur **%s** format specifier ka use karke C ko bataate hain ki ab hum strings ke saath kaam kar rahe hain:

Example

```
char greetings[] = "Hello World!";
printf("%s", greetings);
```

Access Strings

Strings C mein actually arrays hote hain, isliye aap ek string ko uske index number ke through access kar sakte ho, square brackets `[]` ke andar.

Yeh example pehli character (0 index) ko print karega:

Example

```
char greetings[] = "Hello World!";
printf("%c", greetings[0]);
```

Note: Hum `%c` format specifier ka use karte hain single character ko print karne ke liye.

Modify Strings

Agar aapko string mein kisi specific character ka value change karna ho, to aap index number ko refer kar sakte hain, aur single quotes (`''`) ka use karke naya character assign kar sakte hain:

Example

```
char greetings[] = "Hello World!";
greetings[0] = 'J';
printf("%s", greetings);
```

Output:

```
Jello World!
```

Is tarah se aap string ko modify kar sakte ho aur different operations perform kar sakte ho C mein.

C Special Characters

Strings - Special Characters

Strings ko quotes ke andar likhna padta hai, lekin agar aap string mein quotes ka use karte hain, to C ko samajhne mein dikkat ho sakti hai aur error generate ho sakta hai.

Example:

```
char txt[] = "We are the so-called "Vikings" from the north.";
```

Is problem ko avoid karne ka solution hai **backslash escape character** ka use karna.

Backslash (\) escape character special characters ko normal string characters mein convert kar deta hai.

Escape Characters:

Escape Character	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

Example

```
char txt[] = "We are the so-called \"Vikings\" from the north.";
```

The sequence `\'` inserts a single quote in a string:

Example

```
char txt[] = "It\'s alright.";
```

The sequence `\\` inserts a single backslash in a string:

Example

```
char txt[] = "The character \\ is called backslash.";
```

Other Popular Escape Characters:

Escape Character	Result
`\n`	New Line
`\t`	Tab
`\0`	Null

C String Functions

String Functions

C mein kaafi useful string functions hote hain, jo aapko strings par different operations perform karne mein madad karte hain.

In functions ko use karne ke liye, aapko apne program mein `<string.h>` header file ko include karna padta hai:

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

String Length

Agar aapko string ka length pata karna ho, to aap `strlen()` function ka use kar sakte hain:

Example:

```
char alphabet[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
printf("%zu", strlen(alphabet));
```

sizeof VS strlen:

Jab aap `sizeof` aur `strlen` ka comparison karte hain, dono alag tareeke se kaam karte hain. `sizeof` mein string ke saath `\0` (null character) bhi count hota hai, jabki `strlen` string ki actual length batata hai.

Example:

```
char alphabet[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
printf("%zu\n", strlen(alphabet));   // 26
printf("%zu\n", sizeof(alphabet));   // 27
```

Example with allocated space:

```
char alphabet[50] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
printf("%zu\n", strlen(alphabet));   // 26
printf("%zu\n", sizeof(alphabet));   // 50
```

Concatenate Strings

Agar aapko do strings ko combine (concatenate) karna ho, to aap `strcat()` function ka use kar sakte hain:

Example:

```
char str1[20] = "Hello ";
char str2[] = "World!";

// Concatenate str2 to str1 (result is stored in str1)
strcat(str1, str2);

// Print str1
printf("%s", str1);
```

Output:

```
Hello World!
```

Is tarah se aap strings ki length calculate kar sakte hain aur unhe concatenate bhi kar sakte hain C mein!

C User Input

User Input

Aap pehle hi seekh chuke hain ki `printf()` function ka use values ko output karne ke liye hota hai.

Agar aapko user se input lena ho, to aap `scanf()` function ka use karte hain:

Example:

User se ek number input lena:

```
// Create an integer variable that will store the number we get from
the user
int myNum;

// Ask the user to type a number
printf("Type a number: \n");

// Get and save the number the user types
scanf("%d", &myNum);

// Output the number the user typed
printf("Your number is: %d", myNum);
```

Explanation:

- `scanf()` function ko do arguments chahiye: ek format specifier (jaise `%d` integer ke liye) aur ek reference operator (`&myNum`), jo variable ke memory address ko store karta hai.

Tip: Aap aage jaake memory addresses aur functions ke baare mein aur seekhenge.

Multiple Inputs

`scanf()` function ko aap multiple inputs bhi de sakte hain (jaise ek integer aur ek character):

Example:

```
// Create an int and a char variable
int myNum;
char myChar;

// Ask the user to type a number AND a character
printf("Type a number AND a character and press enter: \n");

// Get and save the number AND character the user types
scanf("%d %c", &myNum, &myChar);

// Print the number
printf("Your number is: %d\n", myNum);

// Print the character
printf("Your character is: %c\n", myChar);
```

Take String Input

Aap user se string bhi input kar sakte hain:

Example:

User ka naam input lena:

```
char fullName[30];

printf("Type your full name: \n");
scanf("%s", &fullName);

printf("Hello %s", fullName);

// Type your full name: John Doe
// Hello John
```

Note: Jab aap `scanf()` mein strings ke saath kaam karte hain, to aapko string/array ka size specify karna padta hai (humne example mein `30` liya hai, taaki naam ke liye kaafi space ho). Aur aapko reference operator (`&`) ka use nahi karna padta.

Limitation of `scanf()`

Lekin, `scanf()` function ki kuch limitations bhi hain. Yeh function space (whitespace, tabs, etc.) ko terminating character samajhta hai, isliye yeh sirf ek word tak hi input le pata hai (chahe aap multiple words type karen). Jaise ki:

Example:

```
char fullName[30];

printf("Type your full name: \n");
fgets(fullName, sizeof(fullName), stdin);

printf("Hello %s", fullName);

// Type your full name: John Doe
// Hello John Doe
```

Agar aap `Type your full name: John Doe` type karenge, to program output karega:

```
Hello John
```

Isliye, jab strings ke saath kaam karte hain, hum aksar `fgets()` function ka use karte hain, jo ek line of text ko read karta hai.

fgets() ka use karna

Is function mein aapko kuch arguments specify karne padte hain: string variable ka naam, `sizeof(string_name)`, aur `stdin` (standard input):

Example:

```
char fullName[30];

printf("Type your full name: \n");
fgets(fullName, sizeof(fullName), stdin);

printf("Hello %s", fullName);

// Type your full name: John Doe
// Hello John Doe
```

Output:

```
Type your full name: John Doe
Hello John Doe
```

Is tarah se, aap user se strings, numbers, aur characters ko efficiently input kar sakte hain C mein!

C Memory Address

Memory Address

Jab C mein koi variable create kiya jata hai, to us variable ko ek memory address assign hota hai.

Memory address wo location hota hai jahan variable computer ke memory mein store hota hai.

Jab hum variable ko koi value assign karte hain, to wo value us memory address par store ho jati hai.

Is memory address ko access karne ke liye, aap **reference operator** (`&`) ka use karte hain, aur iska result wo jagah hoti hai jahan variable store hai:

Example:

```
int myAge = 43;
printf("%p", &myAge); // Outputs 0x7ffe5367e044
```

Note:
Memory address hexadecimal form mein hota hai (0x..). Aapko apne program mein same result nahi milega, kyunki yeh depend karta hai ki variable aapke computer par kahan store ho raha hai.

Aapko yeh bhi pata hona chahiye ki `&myAge` ko aksar "pointer" kaha jata hai. Pointer basically ek aisa variable hota hai jo kisi dusre variable ka memory address store karta hai. Pointer values ko print karne ke liye, hum `%p` format specifier ka use karte hain.

C Pointers

Creating Pointers

Aapne pehle chapter mein seekha ki hum reference operator `&` ka use karke kisi variable ka memory address le sakte hain:

Example:

```
int myAge = 43; // an int variable

printf("%d", myAge);  // Outputs the value of myAge (43)
printf("%p", &myAge); // Outputs the memory address of myAge
(0x7ffe5367e044)
```

Pointer ek aisa variable hota hai jo kisi doosre variable ka memory address apni value ke roop mein store karta hai.

A pointer variable usi data type ko point karta hai (jaise `int`), aur ise `*` operator ke saath create kiya jata hai.

Pointer ko us variable ka address assign kiya jata hai jiske saath hum kaam kar rahe hote hain:

Example:

```
int myAge = 43;     // An int variable
int* ptr = &myAge;  // A pointer variable, with the name
ptr, that stores the address of myAge

// Output the value of myAge (43)
```

```
printf("%d\n", myAge);

// Output the memory address of myAge (0x7ffe5367e044)
printf("%p\n", &myAge);

// Output the memory address of myAge with the pointer
(0x7ffe5367e044)
printf("%p\n", ptr);
```

Explanation of Example:

1. Humne ek pointer variable `ptr` create kiya jo `int` type ke variable `myAge` ko point karta hai.
2. Pointer ka type hamesha us variable ke type ke saath match karna chahiye, jaise yahan `int` ka pointer hai (`int* ptr`).
3. `&` operator ka use karke humne `myAge` ka memory address pointer `ptr` ko assign kiya.
4. Ab `ptr` mein `myAge` ke memory address ka value store hai.

Dereferencing

Agar aapko pointer ke through variable ka value dekhna ho, to aap `*` operator ka use karte hain (isey dereference operator kaha jata hai):

Example:

```
int myAge = 43;     // Variable declaration
int* ptr = &myAge;  // Pointer declaration

// Reference: Output the memory address of myAge with the pointer
```

```
(0x7ffe5367e044)
printf("%p\n", ptr);

// Dereference: Output the value of myAge with the pointer (43)
printf("%d\n", *ptr);
```

Note:
Yahan `*` sign thoda confusing ho sakta hai kyunki iska do alag-alag kaam hota hai:

1. Jab declaration mein use hota hai (jaise `int* ptr`), to yeh pointer variable create karta hai.
2. Jab dereference karte hain (jaise `*ptr`), to yeh pointer ke through variable ka value access karta hai.

Pointer Variable Declaration Ke Do Tarike

Aap pointers ko do tareeke se declare kar sakte hain:

```
int* myNum;
```

ya

```
int *myNum;
```

Notes on Pointers

Pointers C ko doosri programming languages jaise Python aur Java se alag banate hain. Pointers C mein kaafi important hote hain, kyunki yeh humein computer ke memory mein data ko manipulate karne ki suvidha dete hain. Isse hum code ko optimize kar sakte hain aur performance improve kar sakte hain. Agar aapko data structures jaise lists, trees, aur graphs ke baare mein pata hai, to aapko yeh samajh mein aayega ki pointers unko implement karne mein kitne useful hote hain. Kabhi-kabhi aapko pointers ka use karna padta hai, jaise files aur memory management ke case mein.

Warning:
Pointers ko handle karte waqt aapko dhyan rakhna padta hai, kyunki galat tarike se pointers ka use karke aap doosre memory addresses mein stored data ko damage kar sakte hain.

C Pointers and Arrays

Pointers & Arrays

Aap pointers ka use arrays ko access karne ke liye bhi kar sakte hain.

Maan lijiye, ek array of integers diya gaya hai:

Example:

```
int myNumbers[4] = {25, 50, 75, 100};
```

Aapne arrays chapter mein seekha ki aap `for` loop ka use karke array ke elements ko loop through kar sakte hain:

Example:

```
int myNumbers[4] = {25, 50, 75, 100};
int i;

for (i = 0; i < 4; i++) {
  printf("%d\n", myNumbers[i]);
}
```

Result:

```
25
50
75
100
```

Ab hum har array element ka value print karne ki jagah, unke memory address ko print karenge:

Example:

```
int myNumbers[4] = {25, 50, 75, 100};
int i;

for (i = 0; i < 4; i++) {
  printf("%p\n", &myNumbers[i]);
}
```

Result:

```
0x7ffe70f9d8f0
0x7ffe70f9d8f4
0x7ffe70f9d8f8
```

```
0x7ffe70f9d8fc
```

Note:

Har element ke memory address ka last number alag hai, aur 4 ka addition ho raha hai.

Yeh isliye hai, kyunki `int` type ka size aam tor par 4 bytes hota hai, yaad rakhein:

Example:

```
// Create an int variable
int myInt;

// Get the memory size of an int
printf("%zu", sizeof(myInt));
```

Result:

```
4
```

To "memory address example" ke hisaab se, aap dekh sakte hain ki compiler har array element ke liye 4 bytes memory reserve karta hai, jo matlab hai ki poora array 16 bytes (4 * 4) memory space leta hai:

Example:

```
int myNumbers[4] = {25, 50, 75, 100};
```

```
// Get the size of the myNumbers array
printf("%zu", sizeof(myNumbers));
```

Result:

```
16
```

Pointers and Arrays ka Relation

To, pointers aur arrays ke beech ka relation kya hai? C mein, array ka naam actually array ke pehle element ka pointer hota hai.

Confused? Chaliye, isse thoda aur samajhte hain, aur phir se apne "memory address example" ka use karte hain.

Array ke pehle element ka memory address, array ke naam ke barabar hota hai:

Example:

```
int myNumbers[4] = {25, 50, 75, 100};

// Get the memory address of the myNumbers array
printf("%p\n", myNumbers);

// Get the memory address of the first array element
printf("%p\n", &myNumbers[0]);
```

Result:

```
0x7ffe70f9d8f0
0x7ffe70f9d8f0
```

Yeh basically yeh matlab hai ki hum arrays ke saath pointers ka use kar sakte hain!

Kaise?

Jab `myNumbers` array ka naam pointer ki tarah kaam karta hai, jo us array ke pehle element ka address hold karta hai, toh aap `*` operator ka use karke us element ko access kar sakte hain:

Example:

```
int myNumbers[4] = {25, 50, 75, 100};

// Get the value of the first element in myNumbers
printf("%d", *myNumbers);
```

Result:

```
25
```

Array ke baaki elements ko access karne ke liye, aap pointer/array ko increment kar sakte hain (+1, +2, etc):

Example:

```
int myNumbers[4] = {25, 50, 75, 100};

// Get the value of the second element in myNumbers
printf("%d\n", *(myNumbers + 1));

// Get the value of the third element in myNumbers
```

```
printf("%d", *(myNumbers + 2));

// and so on..
```

Result:

```
50
75
```

Ya aap isse loop ke through bhi kar sakte hain:

Example:

```
int myNumbers[4] = {25, 50, 75, 100};
int *ptr = myNumbers;
int i;

for (i = 0; i < 4; i++) {
  printf("%d\n", *(ptr + i));
}
```

Result:

```
25
50
75
100
```

Array Elements ko Pointers se Change Karna

Pointers ke saath aap array ke elements ka value bhi change kar sakte hain:

Example:

```
int myNumbers[4] = {25, 50, 75, 100};
// Pehle element ka value 13 se change karo
*myNumbers = 13;

// Dusre element ka value 17 se change karo
*(myNumbers + 1) = 17;

// Pehle element ka value get karo
printf("%d\n", *myNumbers);
// Dusre element ka value get karo

printf("%d\n", *(myNumbers + 1));
```

Result:

```
13
17
```

Is tarah se, pointers aur arrays ko combine karke hum arrays ko access aur manipulate kar sakte hain C mein!