Government Girls Polytechnic Bilaspur

Subject Name: Programming in 'C'
Subject Code: 2022274(022)
Semester : 2nd

Prepared By:
Mr. Nuresh Kumar Dewangan
Lecturer
Department of Computer Science & Engineering

Unit-1
Introduction to 'C ' Programming

Algorithm - Algorithm एक स्पष्ट और सीमित (finite) निर्देशों की श्रृंखला होती है, जिसका उद्देश्य किसी विशेष समस्या का समाधान ढूँढना या किसी कार्य को पूरा करना होता है। Algorithm में input लिया जाता है, और उस पर सुसंगत और निर्धारित (well-defined) तरीके से operations किए जाते हैं, जिसके बाद output प्राप्त होता है।

Algorithm एक step-by-step process है जो हमें किसी समस्या का समाधान ढूँढने के लिए निर्देश देता है। यह एक systematic approach होता है, जिसमें हमें एक-एक कदम उठाकर किसी कार्य को सही तरीके से पूरा करना होता है। Algorithm को इस तरह से लिखा जाता है कि उसे कोई भी व्यक्ति या मशीन (computer) आसानी से समझ सके और उसे फॉलो कर सके।

Algorithm में हर step स्पष्ट (unambiguous) और आसान (simple) होना चाहिए। यदि Algorithm के steps जटिल होंगे या सही से defined नहीं होंगे, तो उससे desired output प्राप्त करना मुश्किल हो सकता है।

एक अच्छे Algorithm में यह जरूरी है कि वह finite (सीमित) हो, यानि वह एक निश्चित संख्या में steps में खत्म हो जाए।

Characteristics of algorithm:

1. **Input (इनपुट):** Algorithm में एक या अधिक inputs होते हैं जिन्हें हम algorithm में उपयोग करते हैं।
2. **Output (आउटपुट):** Algorithm को कम से कम एक output देना चाहिए, जो कि desired result होता है।
3. **Finiteness (सीमितता):** Algorithm को सीमित संख्या में steps में समाप्त होना चाहिए।
4. **Definiteness (स्पष्टता):** हर step का निर्देश स्पष्ट और बिना किसी ambiguity (संदेह) के होना चाहिए।
5. **Effectiveness (प्रभावशीलता):** सभी steps इतने सरल होने चाहिए कि उन्हें manually या मशीन द्वारा execute किया जा सके।

Importance of algorithm:

1. **कोड लिखने से पहले योजना (planning):** Algorithm से पहले हम यह सोचते हैं कि हमें कोड कैसे लिखना है और समस्या का समाधान कैसे मिलेगा।
2. **समस्या का हल व्यवस्थित तरीके से:** Algorithm से हम जटिल समस्याओं को छोटे-छोटे steps में तोड़कर हल कर सकते हैं।
3. **कोड की दक्षता (Efficiency):** Algorithm की सहायता से हम कोड को अधिक efficient बना सकते हैं, जिससे समय और संसाधनों की बचत होती है।
4. **कार्यक्रम में सुधार:** Algorithm से हम पहले ही यह सुनिश्चित कर सकते हैं कि हमारा समाधान सही है और उस पर सुधार करना आसान होता है।

Examples:

1. Find the sum of two numbers

```
Step 1: Start
Step 2: Input A, B
Step 3: Sum = A + B
Step 4: Print Sum
Step 5: Stop
```

2. Check whether a number is positive, negative, or zero

```
Step 1: Start
Step 2: Input number N
Step 3: If N > 0, Print "Positive"
Step 4: Else if N < 0, Print "Negative"
Step 5: Else Print "Zero"
Step 6: Stop
```

3. Check whether a number is even or odd

```
Step 1: Start
Step 2: Input N
Step 3: If N % 2 == 0, Print "Even"
Step 4: Else Print "Odd"
Step 5: Stop
```

4. Find the largest of three numbers

```
Step 1: Start
Step 2: Input A, B, C
Step 3: If A > B and A > C, Print "A is largest"
Step 4: Else if B > C, Print "B is largest"
Step 5: Else Print "C is largest"
Step 6: Stop
```

5. Find the sum of first N natural numbers

```
Step 1: Start
Step 2: Input N
Step 3: Sum = 0
Step 4: For i = 1 to N do
            Sum = Sum + i
Step 5: Print Sum
Step 6: Stop
```

Flowchart - Flowchart एक diagrammatic representation होता है, जो किसी प्रक्रिया या एल्गोरिदम के steps को graphically दिखाता है। यह एक प्रकार का visual tool है जो हमें किसी समस्या के समाधान या कार्य के execution की प्रक्रिया को आसानी से समझने में मदद करता है।

Flowchart में हर step को एक symbol (प्रतीक) द्वारा दर्शाया जाता है, और इन symbols को arrows (तीरें) के माध्यम से जोड़ा जाता है, जो यह बताते हैं कि कार्य का अगला step कौन सा होगा।

Characteristics of flowchart:

1. **Clear Representation:** फ्लोचार्ट किसी प्रक्रिया या एल्गोरिदम को स्पष्ट तरीके से दिखाता है।
2. **Symbols:** फ्लोचार्ट में विशिष्ट symbols का उपयोग होता है, जैसे oval, rectangle, diamond, etc.
3. **Decision Making:** फ्लोचार्ट निर्णय लेने (decision making) की प्रक्रिया को diamond shape से दिखाता है।
4. **Simplicity:** यह आसानी से समझने योग्य होता है और कोई भी व्यक्ति बिना तकनीकी ज्ञान के इसे समझ सकता है।
5. **Time-Saving:** यह किसी प्रक्रिया या एल्गोरिदम को जल्दी और आसानी से visualize करने में मदद करता है।

Symbols of a flowchart:

1. **Flow Line (प्रवाह रेखा)**

- Purpose: Flow line का उपयोग step से step तक के flow को दिखाने के लिए किया जाता है। यह arrows के रूप में होते हैं और यह दिखाते हैं कि प्रक्रिया किस दिशा में जाएगी।
- Symbol: Arrow (तीर)
- Example:
    - Step 1 → Step 2

2. **Terminal (टर्मिनल)**

- Purpose: Terminal symbol process के शुरू और अंत को दर्शाता है। यह Start और Stop दोनों के लिए इस्तेमाल होता है।
- Symbol: Oval or Ellipse (अंडाकार या दीर्घवृत्त)
- Example:
    - Start
    - Stop

3. **Input/Output (इनपुट/आउटपुट)**

- Purpose: Input/Output symbol का उपयोग data input करने और result output करने के लिए किया जाता है। जैसे यूज़र से डेटा लेना या परिणाम दिखाना।
- Symbol: Parallelogram (समांतर चतुर्भुज)
- Example:
    - Input number

- Display result

4. **Processing (प्रोसेसिंग)**

- Purpose: Processing symbol का उपयोग किसी operation (जैसे calculation या data manipulation) को दर्शाने के लिए किया जाता है।
- Symbol: Rectangle (आयत)
- Example:
  - Add A and B
  - Calculate sum

5. **Decision (निर्णय)**

- Purpose: Decision symbol का उपयोग किसी condition (सशर्त निर्णय) को दर्शाने के लिए किया जाता है। यहाँ पर एक question पूछा जाता है और उसके आधार पर दो मार्ग होते हैं: Yes या No।
- Symbol: Diamond (हीरा)
- Example:
  - Is A > B?
  - Is N even?

6. **Connection (संपर्क)**

- Purpose: Connection symbol का उपयोग उस स्थिति में किया जाता है जब flowchart बहुत बड़ा हो और किसी step को अगले पृष्ठ से जोड़ने की आवश्यकता हो। इसे हम link के रूप में समझ सकते हैं।
- Symbol: Circle (वृत्त)
- Example:
  - यह symbol एक step को दूसरे पेज से जोड़ता है।

7. **Off-Page Connectors (ऑफ-पेज कनेक्टर्स)**

- Purpose: Off-page connectors का उपयोग उस स्थिति में किया जाता है जब flowchart का एक भाग किसी अलग पृष्ठ पर होता है। यह symbol page switching को दिखाता है।
- Symbol: A small rectangle with a number or letter inside
- Example:
  - इस symbol का उपयोग एक page से दूसरे page पर जाने के लिए किया जाता है।

Advantages of Flowchart:

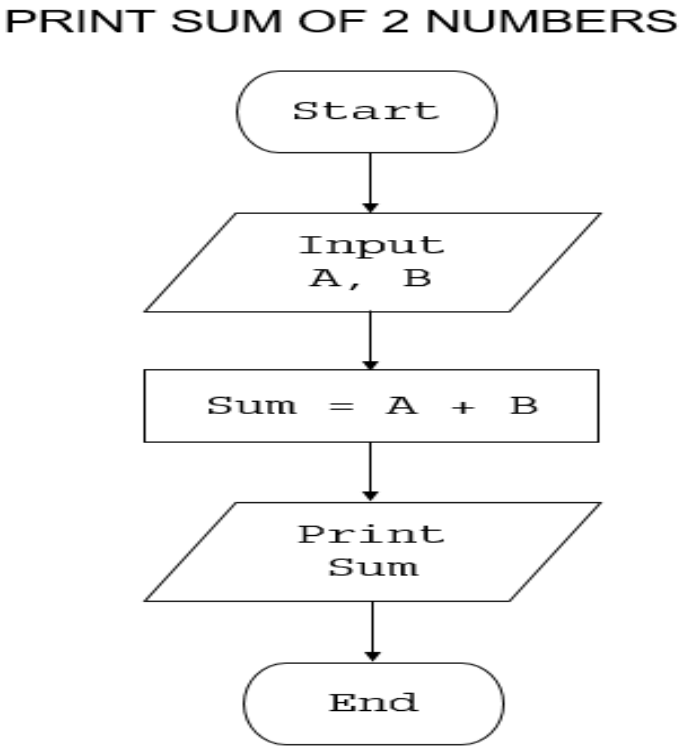
1. **समीक्षा (Review) में आसान:** फ्लोचार्ट में प्रक्रिया को आसानी से समझा जा सकता है और किसी भी गड़बड़ी को जल्दी से पकड़ा जा सकता है।
2. **प्रोब्लम सॉल्विंग में मदद:** जब आपको किसी कार्य को समझने में दिक्कत हो, तो फ्लोचार्ट उसे समझने में मदद करता है।
3. **Efficient Process Design:** फ्लोचार्ट किसी भी कार्य के डिजाइन को सरल और अधिक स्पष्ट बनाता है।
4. **Communication Tool:** टीम के बीच किसी प्रक्रिया को समझाने और discuss करने के लिए फ्लोचार्ट एक बेहतरीन tool है।

Limitations of Flowchart:

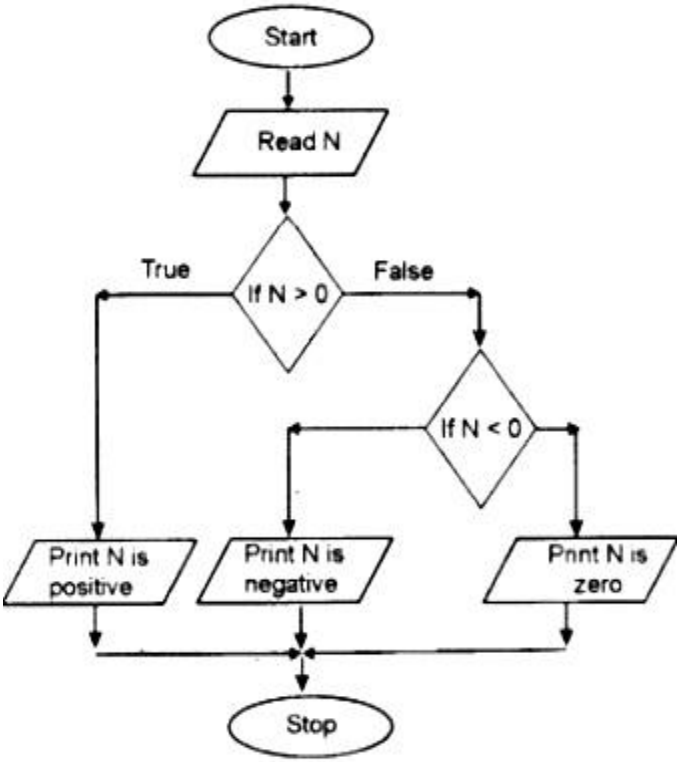
1. **Large Systems में Complexity** - बड़े और जटिल सिस्टम में flowchart को बनाना मुश्किल हो सकता है क्योंकि इसमें बहुत सारे symbols और pages शामिल होते हैं।
2. **Static Representation** - Flowchart केवल static process दिखाता है, dynamic changes या future conditions को नहीं दर्शा सकता।
3. **Complex Algorithms के लिए अनुपयुक्त** - जटिल एल्गोरिदम, जैसे recursion या loops को flowchart में सही से दिखाना कठिन हो सकता है।
4. **Requires Maintenance** - प्रक्रिया में बदलाव होने पर flowchart को update करना पड़ता है, नहीं तो यह पुराना और गलत हो सकता है।
5. **Limited Information** - Flowchart में केवल steps और decisions होते हैं, लेकिन detailed data या parameters को नहीं दर्शा सकता।
6. **Too Many Symbols** - बहुत अधिक symbols और conditions flowchart को overcomplicate कर सकते हैं, जिससे इसे समझना मुश्किल हो जाता है।
7. **Time Consuming** - जटिल processes के लिए flowchart बनाना समय-consuming हो सकता है, और अगर प्रक्रिया बदलती है तो उसे फिर से बनाना पड़ता है।
8. **Sequential Processes तक सीमित** - Flowchart मुख्यतः linear processes को दर्शाने के लिए उपयोगी है, लेकिन parallel processes को दिखाना कठिन हो सकता है।
9. **Ambiguity in Some Cases** - Multiple decisions और conditions होने पर flowchart में ambiguity हो सकती है, जिससे समझने में मुश्किल हो सकती है।
10. **Non-Experts के लिए कठिन** - Non-technical audience के लिए flowchart को समझना मुश्किल हो सकता है, क्योंकि उन्हें symbols और processes का अर्थ नहीं पता होता।

Examples:

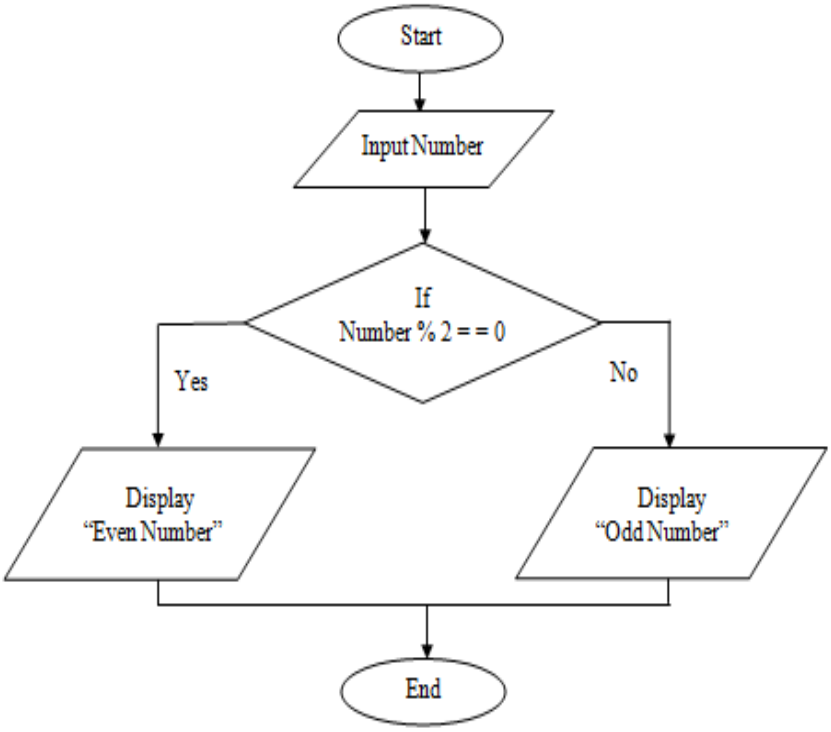
1. Find the sum of two numbers



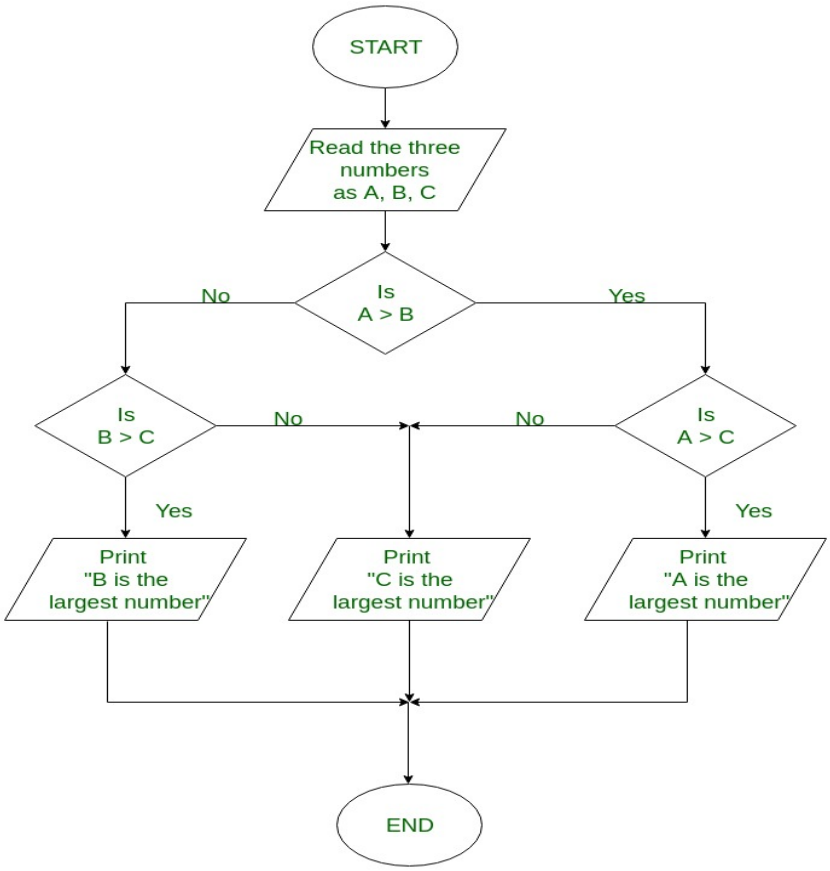
2. Check whether a number is positive, negative, or zero



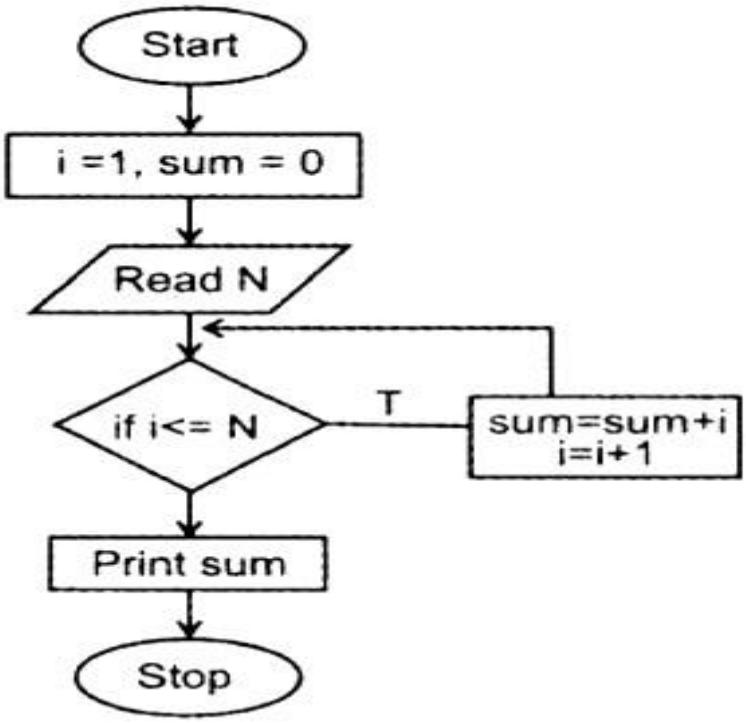
3. Check whether a number is even or odd



4. Find the largest of three numbers



5. Find the sum of first N natural numbers



Basic Structure of C program:

C प्रोग्राम की संरचना साधारणत: कुछ निश्चित हिस्सों से मिलकर बनती है। आइए विस्तार से समझते हैं:

1. **Preprocessor Directives**

- Purpose: प्रीप्रोसेसर निर्देश C प्रोग्राम के सबसे पहले आते हैं। ये प्रोग्राम को कंपाइल करने से पहले कुछ विशेष कार्य करते हैं, जैसे header files को शामिल करना।
- Common Directives:
  - #include <stdio.h> – यह C प्रोग्राम में standard input/output functions के लिए header file को शामिल करता है।
  - #define – constants को define करने के लिए।

Example:

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

2. **Global Declarations**

- Purpose: ये घोषणाएँ प्रोग्राम के किसी भी हिस्से से access की जा सकती हैं। आमतौर पर इसमें global variables और function prototypes होते हैं।

Example:

```
int global_variable = 10;
```

3. **Main Function**

- Purpose: C प्रोग्राम execution का starting point होता है। हर C प्रोग्राम में main() function होना आवश्यक है।
- Syntax:

```
int main() {
   // code
   return 0;
}
```

- Structure:
  - int main() – यह बताता है कि main function का return type int होगा (integer type).
  - return 0; – प्रोग्राम के सफलतापूर्वक खत्म होने को दर्शाता है।

4. **Local Declarations**

- Purpose: ये घोषणाएँ केवल main() function या किसी अन्य user-defined function के भीतर होती हैं। इसमें local variables और temporary data होते हैं।

Example:

```
int a, b;
```

5. **Statements**

- Purpose: इनका उपयोग processing, calculation, या data manipulation के लिए किया जाता है।
- Syntax: Statements प्रोग्राम के logic को implement करती हैं और semicolon (;) से समाप्त होती हैं।

Example:

```
a = 10;  // Assign value 10 to a
b = a + 5;  // Assign value of a + 5 to b
```

6. **Return Statement**

- Purpose: यह प्रोग्राम के execution को समाप्त करता है और return value को operating system को भेजता है।
- Syntax:

```
return 0;
```

C प्रोग्राम की संरचना का उदाहरण

```
#include <stdio.h>  // Preprocessor directive

int global_variable = 10;  // Global declaration

int main() {  // Main function
    int a, b;  // Local declaration

    // Statements
    a = 10;
    b = a + 5;

    printf("Value of b is: %d", b);  // Output statement
    return 0;  // Return statement
}
```

Explanation:

#include <stdio.h>

- यह एक header file है जिसमें input/output functions (जैसे printf, scanf) की definitions होती हैं।
- इसका मतलब है कि इस प्रोग्राम में हम printf() function का उपयोग कर सकते हैं।

int global_variable = 10;

- यह एक global variable है, जो प्रोग्राम के सभी हिस्सों से access किया जा सकता है।
- हालांकि, इस प्रोग्राम में इस variable का उपयोग नहीं किया गया है।

int main() { ... }

- main() function से ही C प्रोग्राम की execution शुरू होती है।

int a, b;

- यहाँ दो local variables (a और b) को declare किया गया है।

a = 10;

- Variable a को value 10 दी गई है।

b = a + 5;

- Variable b को value a + 5, यानी 10 + 5 = 15 दी गई है।

printf("Value of b is: %d", b);

- यह output statement है।
- %d एक format specifier है जो integer value को दिखाने के लिए होता है।
- यह line "Value of b is: 15" को screen पर print करेगी।

return 0;

- यह main function को खत्म करता है और OS को बताता है कि प्रोग्राम सफलतापूर्वक चला है।

Output:

Value of b is: 15

Character set in C:

C programming language में character set उन सभी characters का समूह होता है जिनका उपयोग हम program लिखते समय करते हैं। ये characters C compiler द्वारा recognize किए जाते हैं।

"Character set" वह समूह होता है जिसमें सभी letters, digits, symbols और whitespace characters शामिल होते हैं जिनका उपयोग हम C language के program लिखते समय करते हैं।

C में लिखे गए सभी identifiers, keywords, constants, strings आदि इन्हीं characters से मिलकर बनते हैं। ये characters compiler द्वारा recognize किए जाते हैं।

Types of Character Set in C:

Letters (अक्षर)

- C language में upper-case और lower-case दोनों प्रकार के English letters का उपयोग किया जाता है।
- इन letters का उपयोग variables, functions, identifiers आदि में किया जाता है।

Characters:

- Uppercase: A, B, C, ..., Z
- Lowercase: a, b, c, ..., z

Example:

```
int Age;
float temperature;
```

Digits (अंक)

- Digits का उपयोग numeric values और constants में किया जाता है।
- Digits हमेशा 0 से 9 तक होते हैं।

Characters:

- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Example:

- int roll = 25;

Special Characters (विशेष चिन्ह)

C language में कई प्रकार के special symbols होते हैं जिनका उपयोग punctuation, operators, grouping आदि के लिए किया जाता है।

Common Special Characters Table:

Symbol	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
=	Assignment

Symbol	Meaning
;	Statement Terminator
:	Colon
,	Comma
.	Dot/Decimal Point
()	Parentheses
{}	Braces
[]	Brackets
#	Preprocessor Symbol
&	Address Operator
`	`
!	NOT Operator
< >	Relational Operators

Example:

- int a = b + c; // use of + and =

Whitespace Characters (रिक्त स्थान के कैरेक्टर्स)

Whitespace characters प्रोग्राम में readability बढ़ाने के लिए उपयोग होते हैं और tokens को separate करने में मदद करते हैं।

Types of whitespace characters:

Character	Description
(space)	खाली जगह
\n	newline (नई पंक्ति)
\t	tab space
\r	carriage return

Example:

- int a = 10; // यहाँ space और tab का use है

Escape Sequences (एस्केप कैरेक्टर्स)

Escape characters वो special symbols होते हैं जो backslash \ से शुरू होते हैं और output formatting में काम आते हैं।

Common Escape Sequences:

Escape	Meaning
\n	New line
\t	Tab space
\"	Double quote
\'	Single quote
\\	Backslash
\0	Null character

Example:

- printf("Hello\nWorld"); // output in two lines

Identifiers (पहचानकर्ता)

- Identifiers वो नाम होते हैं जो variables, arrays, functions, आदि को दिए जाते हैं।
- Identifiers में letters, digits और underscore _ का उपयोग हो सकता है परंतु पहला character letter या underscore होना चाहिए।

Example:

- int student_roll_no;

C Tokens:

"C Tokens वो सबसे छोटे meaningful units होते हैं जिनसे मिलकर एक C प्रोग्राम बनता है।"

हर C program अलग-अलग tokens से मिलकर बना होता है। Compiler इन्हीं tokens को पहचानकर program को analyze करता है।

C language में कुल 6 प्रकार के टोकन्स होते हैं:

Keywords

- Keywords ऐसे reserved words होते हैं जिनका meaning C compiler के लिए पहले से तय होता है।
- इनका उपयोग program structure define करने के लिए होता है।
- इन्हें हम variable या function name के रूप में use नहीं कर सकते।
- C में Total 32 Keywords होते हैं ।

Examples:

- int, float, if, else, while, for, return, break, continue

Use in program:

- int age;

यहाँ int एक keyword है।

Identifiers

- Identifiers वो नाम होते हैं जो हम खुद define करते हैं — जैसे variables, functions, arrays आदि के लिए।
- Identifiers meaningful नाम होते हैं जो memory locations को refer करते हैं।

Rules:

- केवल alphabets, digits और underscore (_) allowed हैं।
- पहला character alphabet या underscore होना चाहिए।
- Keywords को identifiers की तरह use नहीं कर सकते।

Examples:

- studentName, total_marks, _value, sum1

Constants

- Constants ऐसी values होती हैं जो program के execution के दौरान change नहीं होतीं।
- ये fixed values होती हैं।

Types of Constants:

Type	Example
Integer Constant	10, -200

Type	Example
Floating Constant	3.14, -0.99
Character Constant	'A', '9'
String Constant	"Hello", "123"

Example in program:

- const int max = 100;

Operators

- Operators वो symbols होते हैं जो values या variables पर operation perform करते हैं जैसे जोड़ना, घटाना, तुलना करना आदि।

Types of Operators:

Type	Symbols
Arithmetic	+, -, *, /, %
Relational	<, >, <=, >=, ==, !=
Logical	&&, ||, !
Assignment	=, +=, -=, *=, /=
Increment/Decrement	++, --
Bitwise	&, `
Conditional	? :

Example:

- a = b + 5;

यहाँ =, + दोनों operators हैं।

Special Symbols

- Special symbols प्रोग्राम की syntax और structure को define करने में मदद करते हैं।

Common Special Symbols:

Symbol	Meaning
;	Statement terminator
{}	Block of code
()	Function call या grouping
[]	Array indexing
#	Preprocessor directive
,	Separator
"	String delimiter
'	Character delimiter

Example:

- printf("Hello");

यहाँ " " और ; special symbols हैं।

Strings (स्ट्रिंग्स)

- String एक group होता है characters का जो double quotes (" ") के अंदर लिखा जाता है।
- C में string को internally एक character array की तरह store किया जाता है और null character \0 से terminate होता है।

Example:

- "Hello World"
- "2025"

Use in program:

- printf("Good Morning");

Example Program with Tokens:

- int a = 10;

Token Type	Token
Keyword	int
Identifier	a
Operator	=
Constant	10
Special Symbol	;

Variable in C:

Variable एक ऐसा नाम होता है जो memory में किसी data को store करने के लिए एक स्थान (location) को represent करता है।इसे हम program के दौरान value assign करने और update करने के लिए use करते हैं।

Variable एक container की तरह होता है जिसमें हम कोई value temporarily रख सकते हैं और जरूरत पड़ने पर उसे change भी कर सकते हैं।

C में Variable के Key Features:

- हर variable का एक नाम (Name) और एक data type होता है।
- Variable की value को runtime पर बदला जा सकता है।
- C language में कोई भी variable use करने से पहले declare करना जरूरी होता है।

Variable Declaration in C

Syntax:

- <data_type> <variable_name>;

या multiple variables एक साथ:

- <data_type> <var1>, <var2>, <var3>;

Examples:

- int age; // integer type variable
- float marks; // float type variable
- char grade; // character type variable
- int a, b, sum; // multiple variable declaration

Variable Initialization - Declaration के समय value देना initialization कहलाता है।

Syntax:

- <data_type> <variable_name> = <value>;

Example:

- int age = 18;
- float pi = 3.14;

Rules for Naming Variables (Variable नाम देने के नियम)

1. नाम में केवल alphabets (A-Z, a-z), digits (0-9) और underscore (_) allowed हैं।
2. नाम का पहला character alphabet या underscore होना चाहिए।
3. Keywords को variable name की तरह use नहीं किया जा सकता।
4. C में variable name case-sensitive होते हैं (e.g., Age और age अलग-अलग हैं)।

Example Program:

```
#include <stdio.h>

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

Output:

```
Sum is: 30
```

Data Types in C (C में डेटा टाइप्स)

“ Data Types वो प्रकार (types) होते हैं जो यह निर्धारित करते हैं कि कोई variable किस तरह का data store करेगा — जैसे integer, character, float आदि। **”**

C programming language में data types का उपयोग memory में सही मात्रा में space allocate करने और data की प्रकृति को परिभाषित करने के लिए किया जाता है।

मुख्य रूप से C में तीन प्रकार के Data Types होते हैं:

1. Primary (Basic) Data Types
2. Derived Data Types
3. User-defined Data Types

1. Primary or Basic Data Types

Data Type	Use / Description	Example Values	Memory (Bytes)
int	Integer values store करने के लिए	10, -5, 1000	2 or 4
float	Decimal (floating point) numbers	3.14, -0.25	4
char	Single character के लिए	'A', 'z', '1'	1
double	Large decimal values के लिए	10.123456	8
void	No value (used in functions)	-	0

2. Derived Data Types

ये data types, basic data types से मिलकर बनते हैं।

Derived Type	Description / Use	Example
Array	एक ही प्रकार के कई values store करने के लिए	int a[10];
Pointer	किसी variable के address को store करता है	int *p;
Function	Code block जो किसी task को perform करता है	void show();
Structure	विभिन्न प्रकार के variables का group	struct student

Derived Type	Description / Use	Example
Union	Memory share करने वाला structure type	union value

3. User-defined Data Types

Programmer खुद ये types define कर सकता है।

Type	Description / Use
struct	अलग-अलग type के variables को group करता है
union	Memory-efficient version of struct
enum	Named integer constants की list बनाता है
typedef	Existing data types को नए नाम से define करता है

1. प्राथमिक डेटा टाइप्स (Primary / Fundamental Data Types)

   यह C के सबसे बुनियादी टाइप होते हैं।

1.1 int

- इसका प्रयोग integer values (whole numbers) को store करने के लिए किया जाता है।
- ये signed और unsigned दोनों हो सकते हैं।
- कोई दशमलव बिंदु नहीं होता।

मेमोरी उपयोग:

- 32-बिट सिस्टम में int की size 4 bytes होती है।
- मान की सीमा: -2,147,483,648 से +2,147,483,647 तक।

उदाहरण:

- int roll_no = 101;
- int year = 2024;

1.2 float

- यह single precision floating point number को store करता है।
- इसमें दशमलव बिंदु होता है।
- scientific calculations में उपयोगी होता है।

मेमोरी उपयोग: 4 bytes
Precision: लगभग 6 अंक तक सटीकता।

उदाहरण:

- float average = 75.5;
- float pi = 3.14;

1.3 double

- यह float से ज्यादा precise होता है।
- बड़ी संख्याएँ या ज्यादा decimal places के लिए उपयोग होता है।

मेमोरी उपयोग: 8 bytes
Precision: 15 से 16 अंक तक।

उदाहरण:

- double value = 123456.7890123;

1.4 char (अक्षर या करैक्टर)

- यह एक single character store करता है।
- character को single quotes में लिखा जाता है।
- internally ASCII value store होती है।

मेमोरी उपयोग: 1 byte
ASCII Range: 0–255

उदाहरण:

- char grade = 'A';
- char symbol = '#';

1.5 void (रिक्त टाइप)

- void का अर्थ है "कुछ नहीं"।
- जब कोई function value return नहीं करता, तब इसे use किया जाता है।

उदाहरण:

- void display(); // कोई return value नहीं

2. व्युत्पन्न डेटा टाइप्स (Derived Data Types)

ये मूल डेटा टाइप्स से बनाए गए होते हैं।

2.1 Array (सरणी)

- एक ही प्रकार के multiple values को एक ही नाम से store करने का माध्यम।
- Memory में contiguous स्थान पर store होता है।

उदाहरण:

- int marks[5] = {90, 85, 75, 80, 95};

2.2 Pointer (सूचक)

- Pointer एक variable होता है जो किसी दूसरे variable के address को store करता है।
- memory handling और dynamic allocation के लिए आवश्यक।

उदाहरण:

- int x = 10;
- int *p = &x;

2.3 Function (कार्य)

- Function एक ऐसा block होता है जो कोई कार्य perform करता है।
- इसे बार-बार call किया जा सकता है।

उदाहरण:

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

3. यूज़र डिफाइन्ड डेटा टाइप्स (User Defined Data Types)

प्रोग्रामर द्वारा परिभाषित विशेष data types होते हैं।

3.1 struct (स्ट्रक्चर)

- Structure में हम अलग-अलग प्रकार के variables को एक unit में store कर सकते हैं।

उदाहरण:

```
struct student {
    int roll;
    char name[20];
    float marks;
};
```

3.2 union (यूनियन)

- Structure जैसा ही होता है लेकिन सभी members एक ही memory space share करते हैं।
- Memory efficient होता है।

उदाहरण:

```
union data {
    int i;
    float f;
};
```

3.3 enum (एनम्यूरेशन)

- यह एक user-defined integer constants की list होती है।

उदाहरण:

```
enum color { RED, GREEN, BLUE };
```

3.4 typedef

- किसी existing type को नया नाम देने के लिए प्रयोग होता है।

उदाहरण:

```
typedef int Number;
Number x = 100;
```

C में Data Type Conversion (डेटा टाइप परिवर्तन)

जब किसी एक प्रकार (data type) के data को दूसरे प्रकार में बदला जाता है, तो इसे Data Type Conversion कहते हैं। C भाषा में यह conversion दो प्रकार का होता है:

1. **Implicit Type Conversion (स्वचालित प्रकारांतरण)**

इसे Type Promotion भी कहते हैं। यह conversion compiler द्वारा स्वतः (automatically) किया जाता है जब दो अलग-अलग data types के operands के बीच कोई expression होता है।

विशेषताएँ:

- छोटे data type को बड़े data type में बदला जाता है।
- Data की precision और safety को ध्यान में रखते हुए conversion होता है।
- कोई programmer intervention (हस्तक्षेप) नहीं चाहिए।

Conversion का क्रम (Hierarchy):

- char -> short -> int -> long -> float -> double -> long double

उदाहरण:

```
int a = 10;
float b = 5.5;
float c;

c = a + b;  // int 'a' को float में convert किया जाएगा
```

Output: c = 15.5

यहाँ a एक int है, लेकिन जब उसे float b के साथ जोड़ा गया, तो compiler ने a को float में बदल दिया (type promotion), जिससे पूरा expression float में हो गया।

2. **Explicit Type Conversion (निर्दिष्ट प्रकारांतरण)**

इसे Type Casting भी कहते हैं। जब programmer स्वयं किसी data को किसी और type में बदलता है, तो उसे explicit conversion कहते हैं।

विशेषताएँ:

- Conversion programmer द्वारा किया जाता है।
- इसके लिए cast operator का उपयोग होता है।

Syntax:

- (new_data_type) expression;

उदाहरण:

```
float f = 10.75;
int a;

a = (int) f;  // float को int में बदला, दशमलव भाग हट गया
```

Output: a = 10

यहाँ float को int में बदलने के कारण 10.75 का दशमलव भाग (0.75) हटा दिया गया।

Note (महत्वपूर्ण बिंदु):

- Implicit conversion में कभी-कभी precision loss हो सकता है।
- Explicit conversion अधिक control देता है, लेकिन गलत cast करने पर data loss या logic error हो सकता है।
- Type conversion का use arithmetic operations, assignment, और function calls में सामान्यतः होता है।

उदाहरण:

```
int x = 5, y = 2;
float result;

result = x / y;  // Implicit Conversion में result = 2.0
```

यहाँ दोनों operand int हैं, तो result भी int होगा, यानी 2। फिर उसे float में convert कर 2.0 बनता है।

अब अगर हम explicit conversion करें:

```
result = (float)x / y;  // अब result = 2.5
```

यहाँ पहले x को float में convert किया गया, जिससे पूरा expression float हो गया।

Operators in C (C के ऑपरेटर)

- Operator वे symbols होते हैं जो variables और values पर कुछ operations perform करते हैं |
- C language में कई प्रकार के operators होते हैं जिनसे arithmetic, comparison, logical और अन्य प्रकार के कार्य किए जा सकते हैं।

1. Arithmetic Operators

यह ऑपरेटर गणितीय क्रियाओं को निष्पादित करते हैं जैसे जोड़, घटाव, गुणा, भाग और शेषफल निकालना।

ऑपरेटर और उनका कार्य:

ऑपरेटर	कार्य	उदाहरण	आउटपुट
+	जोड़ (Addition)	10 + 5	15
-	घटाव (Subtraction)	10 - 5	5
*	गुणा (Multiplication)	10 * 5	50
/	भाग (Division)	10 / 2	5
%	शेषफल (Modulus)	10 % 3	1

Working Mechanism:

- / (Division): जब integer values का division करते हैं तो output integer होता है, और decimal part हटा दिया जाता है।
- % (Modulus): यह operator केवल integer types के साथ काम करता है और division का remainder return करता है।

Example:

```
int a = 10, b = 3;
printf("%d", a / b);   // Output: 3 (Integer division)
printf("%d", a % b);   // Output: 1 (Remainder)
```

2. Relational Operators

इनका उपयोग दो मानों की तुलना करने के लिए किया जाता है। इनका return type हमेशा boolean (0 or 1) होता है।

ऑपरेटर और उनका कार्य:

ऑपरेटर	कार्य	उदाहरण	आउटपुट
==	बराबर है (Equal to)	10 == 5	0 (false)
!=	बराबर नहीं (Not equal to)	10 != 5	1 (true)
>	बड़ा है (Greater than)	10 > 5	1 (true)
<	छोटा है (Less than)	10 < 5	0 (false)
>=	बड़ा या बराबर (Greater or equal to)	10 >= 5	1 (true)
<=	छोटा या बराबर (Less or equal to)	10 <= 5	0 (false)

Working Mechanism:

- == (Equal to): यह operator equality check करता है।
- != (Not equal to): यह check करता है कि दोनों values equal नहीं हैं।
- > (Greater than) और < (Less than): ये operators comparison करते हैं और यह greater या less होने की स्थिति में true return करते हैं।

Example:

```
int a = 10, b = 5;
printf("%d", a > b);   // Output: 1 (true)
printf("%d", a == b);  // Output: 0 (false)
```

3. Logical Operators

इनका उपयोग multiple conditions को combine करने के लिए किया जाता है। ये operators Boolean logic पर काम करते हैं।

ऑपरेटर और उनका कार्य:

ऑपरेटर	कार्य	उदाहरण	आउटपुट
&&	AND – दोनों conditions true होनी चाहिए	1 && 0	0 (false)
||	OR – किसी एक condition का true होना चाहिए	1 || 0	1 (True)
!	NOT – उल्टा (invert the value)	!1	0 (false)

Working Mechanism:

- && (AND): यह तब true return करता है जब दोनों conditions true हों।
- || (OR): यह तब true return करता है जब कम से कम एक condition true हो।
- ! (NOT): यह किसी Boolean value का उल्टा return करता है।

Example:

```
int x = 1, y = 0;
printf("%d", x && y);  // Output: 0 (false) because y is 0
printf("%d", x || y);  // Output: 1 (true) because x is 1
printf("%d", !(x));    // Output: 0 (false) because x is 1
```

4. Assignment Operators

यह ऑपरेटर किसी variable को मान सौंपने के लिए उपयोग होते हैं।

ऑपरेटर और उनका कार्य:

ऑपरेटर	कार्य	उदाहरण
=	मान सौंपना (Assignment)	a = 10;
+=	addition assignment	a += 5; (a = a + 5)
-=	subtraction assignment	a -= 2; (a = a - 2)
*=	multiplication assignment	a *= 3; (a = a * 3)
/=	division assignment	a /= 2; (a = a / 2)
%=	modulus assignment	a %= 3; (a = a % 3)

Working Mechanism:

- =: यह operator किसी variable को एक specific value सौंपने के लिए उपयोग होता है।
- +=, -=, *= आदि operators existing value को modify करते हैं।

Example:

```
int a = 10;
a += 5;  // a = a + 5 => a = 15
printf("%d", a);  // Output: 15
```

5. Bitwise Operators

Bitwise operators का उपयोग binary (0 और 1) स्तर पर काम करने के लिए किया जाता है। यह ऑपरेटर एक-एक बिट पर काम करते हैं।

ऑपरेटर और उनका कार्य:

ऑपरेटर	कार्य	उदाहरण	आउटपुट
&	AND – दोनों bits 1 हों तो 1	5 & 3 (0101 & 0011 = 0001)	1
|	OR – कोई एक bit 1 हो तो 1	5 | 3 (0101 | 0011 = 0111)	7
^	XOR – अगर bits अलग हों तो 1	5 ^ 3 (0101 ^ 0011 = 0110)	6
~	NOT – bit का उल्टा (inverse)	~5 (0101 -> 1010)	-6
<<	Left shift – bits को left shift करना	5 << 1 (0101 << 1 = 1010)	10
>>	Right shift – bits को right shift करना	5 >> 1 (0101 >> 1 = 0010)	2

Explanation:

1. & (AND): यह दोनों bits को 1 होने पर 1 return करता है। अन्यथा 0।
    - Example: 5 & 3 (0101 & 0011) = 0001 → Output: 1
2. | (OR): यह किसी भी एक bit के 1 होने पर 1 return करता है।
    - Example: 5 | 3 (0101 | 0011) = 0111 → Output: 7
3. ^ (XOR): यह तब 1 return करता है जब दोनों bits अलग हों।
    - Example: 5 ^ 3 (0101 ^ 0011) = 0110 → Output: 6
4. ~ (NOT): यह हर bit को उल्टा (invert) कर देता है।
    - Example: ~5 → -6
5. << (Left shift): यह bits को left side में shift करता है। इसका मतलब है, हर bit की जगह को एक position left shift किया जाता है और 0 fill होता है।
    - Example: 5 << 1 (0101 << 1) = 1010 → Output: 10
6. >> (Right shift): यह bits को right side में shift करता है। इसका मतलब है, हर bit की जगह को एक position right shift किया जाता है और sign bit fill होता है।
    - Example: 5 >> 1 (0101 >> 1) = 0010 → Output: 2

Example:

```
int a = 5, b = 3;
printf("%d\n", a & b);  // Output: 1
printf("%d\n", a | b);  // Output: 7
printf("%d\n", a ^ b);  // Output: 6
printf("%d\n", ~a);     // Output: -6
printf("%d\n", a << 1); // Output: 10
```

```
printf("%d\n", a >> 1); // Output: 2
```

6. Increment and Decrement Operators

यह ऑपरेटर किसी variable के मान को बढ़ाने (increment) या घटाने (decrement) के लिए इस्तेमाल होते हैं।

ऑपरेटर और उनका कार्य:

ऑपरेटर	कार्य	उदाहरण	आउटपुट
++a	Pre-increment – पहले बढ़ाए, फिर उपयोग करें	++a	a की value पहले बढ़ती है, फिर उपयोग होती है
a++	Post-increment – पहले उपयोग करें, फिर बढ़ाए	a++	a की value पहले उपयोग होती है, फिर बढ़ती है
--a	Pre-decrement – पहले घटाए, फिर उपयोग करें	--a	a की value पहले घटती है, फिर उपयोग होती है
a--	Post-decrement – पहले उपयोग करें, फिर घटाए	a--	a की value पहले उपयोग होती है, फिर घटती है

Explanation:

- Pre-increment (++a): यह ऑपरेटर पहले value को बढ़ाता है और फिर उस बढ़ी हुई value का उपयोग करता है।
- Post-increment (a++): यह ऑपरेटर पहले value का उपयोग करता है और फिर उसे बढ़ा देता है।
- Pre-decrement (--a): यह ऑपरेटर पहले value को घटाता है और फिर घटित value का उपयोग करता है।
- Post-decrement (a--): यह ऑपरेटर पहले value का उपयोग करता है और फिर उसे घटा देता है।

Example:

```
int a = 5;
printf("%d\n", ++a);  // Output: 6 (Pre-increment)
printf("%d\n", a++);   // Output: 6 (Post-increment, value printed first)
printf("%d\n", a);     // Output: 7 (after post-increment)
```

7. Conditional (Ternary) Operator

यह ऑपरेटर if-else को एक लाइन में compact करता है। इसे ternary operator भी कहा जाता है।

Syntax*:*

```
condition ? expression1 : expression2;
```

यह condition को evaluate करता है:

- यदि condition true है तो expression1 return होगा।
- यदि condition false है तो expression2 return होगा।

Example:

```
int a = 5, b = 10;
int max = (a > b) ? a : b;
printf("%d", max);  // Output: 10, because b is greater
```

Explanation*:*

यह operator conditional if-else logic को compact करता है:

```
if(a > b)
   max = a;
else
   max = b;
```

8. Comma Operator

Comma operator का उपयोग multiple expressions को एक साथ evaluate करने के लिए किया जाता है।

Syntax*:*

```
expression1, expression2, ..., expressionN
```

यह सभी expressions को left to right evaluate करता है और final expression का value return करता है।

Example:

```
int a = 5, b = 10;
int result = (a++, b++, a + b);
printf("%d", result);  // Output: 17
```

Explanation*:*

यह ऑपरेटर expressions को एक के बाद एक execute करता है, लेकिन final value return करता है। यहां पर:

- a++ और b++ पहले execute होते हैं, और अंत में (a + b) का result return होता है।

9. Sizeof Operator

sizeof ऑपरेटर का उपयोग किसी variable या datatype की memory size जानने के लिए किया जाता है।

Syntax*:*

```
sizeof(datatype)
sizeof(variable)
```

Example:

```
int a = 5;
printf("%zu", sizeof(a));  // Output: 4 (assuming int takes 4 bytes)
```

Explanation*:*

यह ऑपरेटर किसी variable या datatype की memory size bytes में return करता है।

10. Pointer Operators

Pointer operators का उपयोग pointers से संबंधित operations करने के लिए किया जाता है।

ऑपरेटर और उनका कार्य:

ऑपरेटर	कार्य	उदाहरण	आउटपुट
&	address-of – किसी variable का address प्राप्त करना	&a	a का address
*	dereference – pointer को dereference करना (value access करना)	*ptr	ptr द्वारा point की गई value

Example:

```
int a = 5;
int *ptr = &a;
printf("%d\n", *ptr);  // Output: 5
```

Explanation*:*

- &: यह operator variable का memory address return करता है।
- *: यह operator pointer को dereference करता है और उसकी value access करता है।

11. Typecast Operator

Typecasting का मतलब है, किसी variable की data type को explicitly बदलना।

Syntax*:*

```
(type) variable
```

Example:

```
float f = 10.5;
int i = (int) f;  // typecasting float to int
printf("%d", i);  // Output: 10
```

Explanation*:*

यह ऑपरेटर explicit type conversion करता है और एक type से दूसरे type में value को बदलता है।

12. Member Access Operator

. (dot) और -> operators का उपयोग structure और pointer to structure के members तक पहुँचने के लिए किया जाता है।

Example:

```
struct Student {
    int age;
    float marks;
};

struct Student s1;
s1.age = 20;          // Using dot operator
printf("%d", s1.age); // Output: 20
```

Explanation*:*

- . : structure के instance से member तक पहुँचने के लिए।
- -> : structure pointer से member तक पहुँचने के लिए।

13. sizeof Operator

sizeof ऑपरेटर का उपयोग variable या datatype के size को bytes में प्राप्त करने के लिए किया जाता है।

Example:

```
int a = 10;
printf("%zu", sizeof(a));  // Output: 4 (Size of int is typically 4 bytes)
```

C Input-Output Function

1. **printf() Function**

printf() function को हम output देने के लिए उपयोग करते हैं। यह data को screen पर print करता है।

- Header File: #include <stdio.h>
- Return Type: int

यह function successfully print होने वाले characters की संख्या को return करता है।

Syntax:

```
int printf("format string", argument1, argument2, ...);
```

Arguments:

- इसका पहला argument हमेशा double quotes (" ") में होता है — जिसे format string कहा जाता है।
- इसके बाद वाले arguments वो variables होते हैं जिनकी value print करनी होती है।
- यह function multiple arguments ले सकता है।

Format Specifiers:

Specifier	Data Type
%d	Integer
%f	Float
%c	Character
%s	String
%lf	Double

Example:

```
#include <stdio.h>

int main() {
   int a = 10;
   float b = 5.75;
   char c = 'A';

   printf("Integer = %d\n", a);
   printf("Float = %.2f\n", b);
   printf("Character = %c", c);

   return 0;
}
```

Output:

```
Integer = 10
Float = 5.75
Character = A
```

2. **scanf() Function**

यह function user से input लेने के लिए प्रयोग किया जाता है।

- Header File: #include <stdio.h>
- Return Type: int

यह function सफलतापूर्वक input लिए गए variables की संख्या return करता है।

Syntax:

```
int scanf("format string", &variable1, &variable2, ...);
```

Arguments:

- पहला argument double quotes में format string होता है।
- बाकी arguments उन variables के addresses होते हैं जिन्हें input के रूप में store करना होता है (इसलिए & sign जरूरी है)।
- यह भी multiple arguments ले सकता है।

Example:

```
#include <stdio.h>

int main() {
    int age;
    float marks;

    printf("Enter age and marks: ");
    scanf("%d %f", &age, &marks);

    printf("Age: %d, Marks: %.2f", age, marks);
    return 0;
}
```

Output:

```
Enter age and marks: 21 84.5
Age: 21, Marks: 84.50
```

3. **getchar() Function**

यह function keyboard से एक single character input लेता है और उसे return करता है।

- Header File: #include <stdio.h>
- Return Type: int

हालांकि यह character return करता है, लेकिन इसकी return type int होती है क्योंकि यह character का ASCII value return करता है।

Syntax:

```
int getchar(void);
```

Arguments:

कोई arguments नहीं लेता।

Example:

```
#include <stdio.h>

int main() {
   char ch;

   printf("Enter a character: ");
   ch = getchar();

   printf("You entered: %c", ch);
   return 0;
}
```

Output:

```
Enter a character: A
You entered: A
```

4. **putch() Function**

यह function एक character को output के रूप में screen पर दिखाता है।

- Header File: #include <conio.h>
- Return Type: int

यह function वो character return करता है जिसे print किया गया है।

Syntax:

```
int putch(char character);
```

Arguments:

- सिर्फ एक argument लेता है — character type

Example:

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

int main() {
   char ch = 'Z';
   putch(ch);
```

```
    return 0;
}
```
Output:
```
Z
```

5. **getch() Function**

यह function keyboard से एक character input करता है, लेकिन वो screen पर show नहीं होता।

- Header File: #include <conio.h>
- Return Type: int

यह function ASCII value return करता है।

Syntax:

```
int getch(void);
```

Arguments:

कोई argument नहीं लेता।

Example:

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

int main() {
    char ch;
    printf("Press any key to continue...");
    ch = getch();  // character input (invisible)
    return 0;
}
```
Output:

Press any key to continue...

6. putchar() Function

putchar() function का उपयोग किसी एक single character को screen पर print करने के लिए किया जाता है। यह standard output device (जैसे कि monitor) पर character को display करता है।

- Header File: #include <stdio.h>
- Return Type: int

यह function उस character का ASCII value return करता है जिसे output किया गया है।

Syntax:

```
int putchar(int character);
```

Arguments:

एक argument लेता है — जो integer के रूप में दिया गया character होता है (ASCII value के रूप में)।

Example:

```
#include <stdio.h>

int main() {
    char ch = 'M';
    putchar(ch);  // prints 'M'
    return 0;
}
```

Output:

```
M
```

Conclusion Summary Table:

Function	Purpose	Header File	Return Type	Arguments	Input/Output	Notes
printf()	Print data to screen	stdio.h	int	Multiple	Output	Format string in double quotes
scanf()	Take input from user	stdio.h	int	Multiple	Input	Use & for variable addresses
getchar()	Input single character	stdio.h	int	None	Input	Returns ASCII value of character
putch()	Print single character	conio.h	int	One	Output	Used for displaying a single character
getch()	Take hidden character input	conio.h	int	None	Input	Commonly used to pause program
putchar()	Print single character	stdio.h	int	One	Output	Returns ASCII value of printed character

Unit-2
Decision Making and Branching Statement

Control Flow Statements in C

C भाषा में Control Flow Statements का प्रयोग प्रोग्राम की execution की दिशा (flow of control) को नियंत्रित करने के लिए किया जाता है। ये statements यह तय करते हैं कि प्रोग्राम के कौन से भाग कब execute होंगे। इनके बिना कोई भी प्रोग्राम linear तरीके से चलता है — यानि एक के बाद एक statement execute होता है। लेकिन अधिक complex और logical प्रोग्राम बनाने के लिए control flow statements अत्यंत आवश्यक होते हैं।

C में Control Flow Statements को मुख्यतः तीन श्रेणियों में बाँटा गया है:

1. Decision-Making Statements

इनका उपयोग प्रोग्राम में किसी *शर्त (condition)* के आधार पर विभिन्न कार्यों को करने के लिए किया जाता है। जब हमें किसी condition के अनुसार निर्णय लेना होता है — जैसे कि कोई block execute करना है या नहीं, तब decision-making statements उपयोगी होते हैं। यह control flow को conditional बना देते हैं।

इस प्रकार के statements में प्रोग्राम किसी condition को evaluate करता है (true/false) और उसी के आधार पर कोई विशेष code block execute करता है। अगर condition false होती है, तो या तो कुछ नहीं होता या कोई alternate block execute होता है।

इस श्रेणी में आने वाले statements:

- if statement
- if...else statement
- else if ladder
- nested if statement
- switch statement

2. Looping Statements

Looping statements का उपयोग तब किया जाता है जब किसी task को बार-बार दोहराने (repeat) की आवश्यकता होती है। यह repetition तब तक होती है जब तक कोई विशेष condition satisfy होती है।

इन statements की मदद से हम एक ही block of code को multiple times execute कर सकते हैं, जिससे code में redundancy घटती है और efficiency बढ़ती है। इसमें एक initialization, condition checking और update (increment/decrement) शामिल होते हैं।

Looping से प्रोग्राम को automation की तरह काम करने में सहायता मिलती है — जैसे कि user से बार-बार input लेना, या एक table print करना आदि।

इस श्रेणी में आने वाले statements:

- for loop
- while loop
- do...while loop

3. Jumping Statements

Jumping statements प्रोग्राम के flow को एक जगह से दूसरी जगह पर तुरंत transfer कर देते हैं। ये control को या तो loop से बाहर निकालते हैं, या अगली iteration पर भेजते हैं, या फिर किसी fixed label पर भेजते हैं।

इनका उपयोग मुख्यतः loop को बीच में से रोकने (terminate करने), किसी iteration को skip करने या अचानक किसी अन्य भाग में jump करने के लिए किया जाता है। यद्यपि इनका प्रयोग संयमपूर्वक करना चाहिए क्योंकि इससे code की readability और maintainability पर असर पड़ सकता है।

इस श्रेणी में आने वाले statements:

- break statement
- continue statement
- goto statement
- return statement

Decision Making Statements in C

C भाषा में Decision Making Statements का उपयोग तब किया जाता है जब प्रोग्राम को किसी शर्त (Condition) के आधार पर यह निर्णय लेना होता है कि कौन-सा निर्देश (statement) execute किया जाए और कौन-सा नहीं।

प्रोग्राम को स्मार्ट बनाने के लिए यह ज़रूरी होता है कि वह अलग-अलग conditions पर अलग-अलग actions ले सके। Decision making statements प्रोग्राम को ऐसी decisions लेने की सुविधा प्रदान करते हैं।

विशेषता (Features):

- यह statements प्रोग्राम को dynamic बनाते हैं।
- User input या runtime data के आधार पर कार्य करने की क्षमता देते हैं।
- Different logical branches बनाकर program की flow को control किया जाता है।

Decision Making Statements के प्रकार:

C भाषा में निम्नलिखित decision making statements होते हैं:

1. if Statement:

   यह सबसे basic decision making structure है। इसमें एक condition को check किया जाता है। यदि condition सही होती है (true), तो associated block of code run होता है। यदि condition गलत होती है (false), तो वह block skip कर दिया जाता है।

2. if...else Statement:

   इस structure में दो भाग होते हैं — एक if block और एक else block। अगर दी गई condition सही होती है, तो if block run होता है, अन्यथा else block run होता है। इसका उपयोग binary decision लेने में होता है।

3. else if Ladder:

   जब एक से अधिक conditions को क्रम से check करना हो, तो else if ladder का उपयोग किया जाता है। इसमें conditions को top से bottom तक check किया जाता है और जैसे ही कोई condition true होती है, वही block execute होता है और बाकी conditions check नहीं की जाती।

4. Nested if Statement:

   जब किसी condition के अंदर और conditions check करनी हों, तो nested if का उपयोग किया जाता है। इसमें एक if या else block के भीतर दूसरा if structure मौजूद होता है। इसका उपयोग complex decision logic को implement करने के लिए किया जाता है।

5. switch Statement:

   यह एक multi-way branching statement है, जो किसी एक expression के result के आधार पर multiple fixed values से match करता है। हर value को एक case कहा जाता है। जब expression किसी case value से match करता है, तो उस block के अंदर का code run होता है। अगर कोई भी case match नहीं करता, तो default block execute होता है।

if Statement:

if statement C प्रोग्रामिंग की सबसे मूलभूत (basic) decision making स्टेटमेंट होती है। इसका उपयोग किसी एक condition को जांचने के लिए किया जाता है। यदि दी गई condition सही (true) होती है, तो संबंधित block of code execute होता है। यदि condition गलत (false) होती है, तो वह block skip कर दिया जाता है।

Syntax:

```
if (condition) {

   // statements to be executed if condition is true

}
```

- **if** एक कीवर्ड है।
- **condition** कोई ऐसा expression होता है जो true या false return करता है।
- अगर condition **true** होती है, तो {} के अंदर का कोड चलेगा।
- अगर condition **false** होती है, तो कुछ भी नहीं होगा (statement skip हो जाएगा)।

विशेषताएँ (Features):

- केवल एक condition को evaluate करता है।
- केवल true होने पर ही block execute करता है।
- simple decision लेने के लिए प्रयोग किया जाता है।

Example:

```
#include <stdio.h>

int main() {
   int number = 10;

   if (number > 0) {
      printf("Number is positive.");
   }

   return 0;
}
```

Output:

```
Number is positive.
```

if-else Statement:

जब हमें किसी condition के true और false दोनों ही case में अलग-अलग instructions execute करवानी हो, तब हम if-else statement का प्रयोग करते हैं।
यह decision making का दूसरा level है, जो हमें दो रास्तों में से एक को चुनने की सुविधा देता है।

Syntax:

```
if (condition) {
   // Condition true होने पर यह block execute होगा
} else {
   // Condition false होने पर यह block execute होगा
```

```
}
```

- if condition को check करता है।
- यदि condition true है, तो पहला block चलेगा।
- यदि condition false है, तो else block चलेगा।

विशेषताएँ (Features):

- दो संभावनाओं में से एक को select करता है।
- program के flow को condition के आधार पर control करता है।
- useful होता है जब एक decision के दो alternative हों।

Example:

```
#include <stdio.h>


int main() {
   int num = -5;


   if (num >= 0) {
      printf("Number is positive or zero.");
   } else {
      printf("Number is negative.");
   }


   return 0;
}
```

Output:

```
Number is negative.
```

Nested if-else Statement:

जब एक if या else के block के अंदर फिर से कोई if-else statement लिखा जाता है, तो उसे nested if-else कहा जाता है।

- इसका उपयोग तब किया जाता है जब हमें एक से अधिक स्तरों पर निर्णय लेने हों।
- यानी, किसी एक condition के true होने पर हम एक और condition check करते हैं।

Syntax (संरचना):

```
if (condition1) {
   if (condition2) {
      // block executes if both condition1 and condition2 are true
   } else {
      // executes if condition1 is true and condition2 is false
   }
} else {
   // executes if condition1 is false
}
```

विशेषताएँ (Features):

- एक decision के अंदर दूसरा decision लेने की सुविधा देता है।
- जटिल परिस्थितियों में उपयोगी होता है, जहाँ कई स्तरों पर जाँच करनी होती है।
- Code थोड़ा complex हो सकता है, लेकिन structured logic के लिए आवश्यक होता है।

Example:

```
#include <stdio.h>

int main() {
   int marks = 85;

   if (marks >= 50) {
      if (marks >= 75) {
         printf("Distinction");
      } else {
```

```
            printf("Pass");
        }
    } else {
        printf("Fail");
    }

    return 0;
}
```

Output:

```
Distinction
```

else-if Ladder Statement:

जब हमें multiple conditions check करनी होती हैं — यानी एक के बाद एक कई विकल्पों में से किसी एक को चुनना होता है — तब हम else-if ladder का प्रयोग करते हैं।

- यह sequential रूप से conditions को check करता है।
- जैसे ही कोई एक condition true होती है, उसका block execute होता है और बाकी conditions check नहीं होतीं।

Syntax (संरचना):

```
if (condition1) {
    // block for condition1
} else if (condition2) {
    // block for condition2
} else if (condition3) {
    // block for condition3
}
...
else {
    // default block if no condition matches
```

```
}
```

विशेषताएँ (Key Features):

- एक से अधिक conditions को check करने के लिए उपयोगी।
- Code को compact और readable बनाता है।
- जैसे ही कोई condition true होती है, आगे की conditions को skip कर दिया जाता है।
- अंत में else block एक default case की तरह काम करता है।

Example:

```
#include <stdio.h>

int main() {
    int marks = 68;

    if (marks >= 90) {
        printf("Grade A");
    } else if (marks >= 75) {
        printf("Grade B");
    } else if (marks >= 60) {
        printf("Grade C");
    } else if (marks >= 40) {
        printf("Grade D");
    } else {
        printf("Fail");
    }

    return 0;
}
```

☑ Output:

```
Grade C
```

switch Statement:

जब किसी variable के मान (value) के आधार पर हमें multiple cases को handle करना होता है, तब हम switch statement का उपयोग करते हैं।

- यह एक multi-way decision making structure है।
- if-else की तरह conditions के आधार पर branching करता है,
- लेकिन switch को अधिक structured और readable माना जाता है, खासकर जब एक variable के multiple fixed values को check करना हो।

Syntax (संरचना):

```
switch(expression) {
   case value1:
      // statements
      break;
   case value2:
      // statements
      break;
   ...
   default:
      // default statements
}
```

- **switch(expression):** यहाँ expression एक constant या integer/character type का होना चाहिए।
- **case:** प्रत्येक case किसी specific value को check करता है। यदि expression उस value के बराबर होता है, तो उस case का code execute होता है।
- **break:** एक case complete होने के बाद control को switch block के बाहर ले जाता है। यदि break नहीं लिखा गया, तो fall-through हो सकता है यानी नीचे के सारे case execute हो सकते हैं।
- **default:** यदि कोई भी case match नहीं करता, तो यह block execute होता है। यह optional है।

विशेषताएँ (Key Points):

- सिर्फ integer और character values के साथ काम करता है।
- float या double values के साथ काम नहीं करता।
- ज़्यादा structured होता है जब multiple constant values को check करना हो।
- break बहुत महत्वपूर्ण होता है — उसके बिना सभी नीचे के cases execute हो सकते हैं।

Example:

```
#include <stdio.h>

int main() {
   int day = 3;

   switch(day) {
      case 1:
         printf("Monday");
         break;
      case 2:
         printf("Tuesday");
         break;
      case 3:
         printf("Wednesday");
         break;
      case 4:
         printf("Thursday");
         break;
      default:
         printf("Invalid Day");
   }

   return 0;
}
```

Output:

Wednesday

goto Statement:

goto statement C में एक control flow statement है जो किसी specific location (label) पर program control को transfer करता है।
यह unconditional branching के लिए उपयोगी होता है, यानी किसी भी condition के बिना program के किसी अन्य भाग पर control transfer कर देता है।
goto का उपयोग तब किया जाता है जब हम program में किसी खास location पर सीधे कूदना चाहते हैं, जैसे loop से बाहर निकलने के लिए, या error handling के लिए।

Syntax (संरचना):

```
goto label;


label:
  // statements
```

- **goto label;:** यह label नामक स्थान (location) पर control transfer करता है।
- **label:** एक identifier होता है जो program में एक स्थान (address) को represent करता है।

विशेषताएँ (Key Features):

- goto किसी भी जगह पर control को transfer कर सकता है, लेकिन यह program को समझने में मुश्किल बना सकता है।
- goto का उपयोग तब करना चाहिए जब कोई अन्य तरीका काम न करे, क्योंकि यह program की readability को प्रभावित करता है।
- goto से control को किसी भी स्थान पर भेजा जा सकता है जो label द्वारा defined हो।
- अगर हम goto का अत्यधिक उपयोग करते हैं, तो program में spaghetti code हो सकता है, जिसमें control flow समझना मुश्किल हो जाता है।

goto Statement का उपयोग कब करें? (When to Use)

- Error handling के दौरान, जैसे किसी condition को fail होते ही program के end पर चले जाना।
- Loop या अन्य control structures को skip करने के लिए।
- जहां multiple conditions के साथ decision making होती है और हम program flow को jump कराना चाहते हैं।

Example:

```
#include <stdio.h>


int main() {
   int i = 0;


   // Label to jump to
   start:
      if(i < 5) {
         printf("%d ", i);
         i++;
         goto start;  // Jumping back to the start label
      }
   return 0;
}
```

Output:

```
0 1 2 3 4
```

?: Operator:

?: operator, जिसे ternary operator भी कहा जाता है, C programming language में एक conditional operator है। यह एक shorthand method है जिसका उपयोग if-else statement को compact रूप में लिखने के लिए किया जाता है।

यह एक ternary (तीन operands वाला) operator है, जो तीन parts में बाँटा जाता है:

Syntax (संरचना):

condition ? expression1 : expression2;

- **condition:** यह एक expression होता है जो true या false value return करता है।
- **expression1:** यदि condition true होता है, तो यह expression execute होता है।
- **expression2:** यदि condition false होता है, तो यह expression execute होता है।
- यह conditional operator if-else के रूप में काम करता है, लेकिन short form में।

विशेषताएँ (Key Features):

- Shorthand for if-else: यह if-else statement को एक line में compact तरीके से लिखने के लिए उपयोगी है।
- Ternary (Three parts): इसमें तीन parts होते हैं — condition, true part, और false part।
- Return Type: यह उस value का return type होगा जो expression1 या expression2 का type हो।

Example:

```
#include <stdio.h>

int main() {
   int a = 10, b = 20;
   int result;

   // Using the ternary operator
   result = (a > b) ? a : b;  // If a > b, result will be a, otherwise b

   printf("The greater number is: %d", result);

   return 0;
}
```

Output:

```
The greater number is: 20
```

Unit-3
Structured Loop Control Statement

Looping Statement (Iteration Statements):

C language में looping statements का उपयोग किसी statement या block of code को बार-बार (repeatedly) execute करने के लिए किया जाता है, जब तक कोई विशेष condition true रहती है। इस प्रक्रिया को Iteration कहा जाता है।

जब किसी कार्य को कई बार करना हो (जैसे किसी list को print करना, sum निकालना, table बनाना आदि), तब एक ही block को कई बार execute करने के लिए loop का उपयोग किया जाता है।

Types of Looping Statements in C:

C में निम्नलिखित तीन प्रकार के looping statements होते हैं:

श्रेणी (Category)	Statement Name	प्रकार (Type)
Entry-Controlled Loop	for loop	पहले condition check होती है
	while loop	पहले condition check होती है
Exit-Controlled Loop	do-while loop	बाद में condition check होती है

Loop के Execution को Control करने वाले Statement:

Loop के execution को modify या control करने के लिए नीचे दिए गए loop control statements का उपयोग किया जाता है:

Statement	कार्य (Purpose)
break	Loop को तुरंत रोकने के लिए (terminate the loop early)
continue	current iteration को छोड़ कर next iteration पर जाने के लिए

for Loop

C Programming में for loop का उपयोग किसी कार्य को एक निश्चित संख्या में दोहराने (repeating a task for a fixed number of times) के लिए किया जाता है। यह एक entry-controlled loop होता है, यानी पहले condition check होती है, फिर statement execute होता है।

Syntax:

```
for(initialization; condition; increment/decrement) {
   // Loop body - statements to be repeated
}
```

Components:

Initialization**:** यह loop की शुरुआत में एक बार होता है। इसमें loop control variable को initialize किया जाता है।
जैसे int i = 0;

Condition**:** यह Boolean expression होता है। जब तक यह condition true रहती है, loop चलता रहता है।
जैसे i < 5

Increment/Decrement**:** हर iteration के बाद control variable को update किया जाता है।
जैसे i++ या i--

Working:

- सबसे पहले initialization होता है।
- फिर condition check होती है:
- अगर condition true है → Loop body execute होती है।
- अगर false है → Loop terminate हो जाता है।
- Loop body के बाद increment/decrement होता है।
- फिर से condition check होती है, और यह process तब तक repeat होता है जब तक condition false न हो जाए।

Example 1: Print numbers from 1 to 5

```
#include <stdio.h>

int main() {

   int i;

   for(i = 1; i <= 5; i++) {

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

   }

   return 0;

}
```

Output:

1

2

3

4

5

Example 2: Reverse counting from 5 to 1

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

Output:

```
5
4
3
2
1
```

Important Points:

- for loop का उपयोग तब करना चाहिए जब iteration की संख्या पहले से ज्ञात हो।
- Initialization, condition, और increment/decrement को एक ही line में manage किया जाता है।
- सभी 3 expressions (initialization; condition; increment) optional होते हैं। अगर हम उन्हें छोड़ देते हैं तो infinite loop बन सकता है।

```
for( ; ; ) {
    printf("Infinite Loop\n");
}
```

while Loop

C programming में while एक entry-controlled loop है। इसका उपयोग तब किया जाता है जब हमें नहीं पता कि loop कितनी बार चलेगा, लेकिन condition के true होने तक वह लगातार execute होता रहेगा।

जब तक दी गई condition true रहती है, तब तक loop के अंदर का code बार-बार execute होता है।

Syntax:

```
while(condition) {
   // Loop body - Statements to be repeated
}
```

Components:

Condition (शर्त) – एक boolean expression होता है।
यदि यह true है → loop चलता रहेगा।
यदि false है → loop terminate हो जाता है।

Loop Body – वह set of statements जो condition true रहने पर बार-बार चलेगा।

Update Statement – loop control variable को update करने का कार्य हम manually करते हैं (जैसे i++ या i--), वरना infinite loop हो सकता है।

Working of while loop:

- सबसे पहले condition check होती है।
- अगर condition true है, तो loop body execute होती है।
- उसके बाद फिर से condition check होती है।
- यह process तब तक दोहराई जाती है जब तक condition false न हो जाए।
- जैसे ही condition false होती है, loop terminate हो जाता है।

Example 1: Print numbers from 1 to 5

```
#include <stdio.h>
int main() {
   int i = 1;
   while(i <= 5) {
      printf("%d\n", i);
```

```
        i++;
    }
    return 0;
}
```

Output:

1

2

3

4

5

Example 2: Print even numbers from 2 to 10

```
#include <stdio.h>
int main() {
    int i = 2;
    while(i <= 10) {
        printf("%d ", i);
        i = i + 2;
    }
    return 0;
}
```

Output:

2 4 6 8 10

Example 3: Infinite Loop

```
#include <stdio.h>
int main() {
    int i = 1;
    while(i <= 5) {
```

```
        printf("%d ", i);
        // i++ को छोड़ दिया गया है, जिससे loop कभी खत्म नहीं होगा
    }
    return 0;
}
```

यह एक infinite loop बन जाएगा क्योंकि i++ missing है।

Important Points:

- while loop में condition पहले check होती है, इसलिए इसे entry-controlled loop कहते हैं।
- यदि condition शुरू में false है, तो loop body एक बार भी execute नहीं होती।
- Update (increment/decrement) statement को loop body के अंदर manually लिखना जरूरी होता है।
- यह loop उन conditions के लिए उपयुक्त है जहाँ repetition की संख्या पहले से ज्ञात नहीं होती।

do-while Loop

C programming में do-while एक exit-controlled loop है। इसका उपयोग तब किया जाता है जब हमें loop body को कम से कम एक बार execute करना हो, भले ही condition false हो।

इस loop में पहले loop body चलती है और उसके बाद condition check होती है। यदि condition true हुई, तो loop दुबारा चलेगा।

Syntax:

```
do {
   // Loop body - statements to execute
} while(condition);
```

Note: while(condition); के बाद semicolon (;) लगाना जरूरी होता है।

Components:

- Loop Body: वो statement जो पहली बार बिना condition check किए execute होंगे।
- Condition: Boolean expression जो यह तय करता है कि loop दुबारा चलेगा या नहीं।
- Update Statement: Condition को false करने के लिए loop control variable को update करना पड़ता है।

Working (कार्यप्रणाली):

- सबसे पहले loop body के statements execute होंगे।
- उसके बाद condition check की जाएगी।
- यदि condition true है, तो loop फिर से चलेगा।
- यदि condition false है, तो program loop से बाहर आ जाएगा।

कब Use करते हैं:

जब हमें किसी operation को कम से कम एक बार जरूर चलाना हो, फिर चाहे condition false ही क्यों न हो।

जैसे – user input लेना, menu driven program, etc.

Example 1: Print 1 to 5 using do-while

```
#include <stdio.h>

int main() {
    int i = 1;
    do {
        printf("%d\n", i);
        i++;
    } while(i <= 5);
    return 0;
}
```

Output:

```
1
2
3
4
5
```

Example 2: Loop runs even if condition is false initially

```
#include <stdio.h>
int main() {
    int i = 10;
    do {
        printf("Value is: %d\n", i);
        i++;
    } while(i < 5);
    return 0;
}
```

Output:

```
Value is: 10
```

ध्यान दें कि condition false थी (i < 5) फिर भी output एक बार आया।

Infinite Loop Example:

```
#include <stdio.h>
int main() {
    int i = 1;
    do {
        printf("%d ", i);
        // i++ नहीं लिखा गया है
    } while(i <= 5);
    return 0;
}
```

इस code में loop कभी खत्म नहीं होगा → Infinite loop बन जाएगा।

Key Points:

- do-while loop एक exit-controlled loop है।
- Loop body पहले execute होती है, फिर condition check होती है।
- इसलिए, loop body कम से कम एक बार जरूर चलती है, चाहे condition false हो।
- Update statements (जैसे i++) को manually add करना पड़ता है।

Difference between while and do-while loop:

Criteria / Point of Difference	while Loop	do-while Loop
Definition (परिभाषा)	Entry-controlled loop — पहले condition check, फिर body execute होती है।	Exit-controlled loop — पहले body execute, फिर condition check होती है।
Condition Check	Loop शुरू करने से पहले condition check होती है।	Loop body पहले चलती है, फिर condition check होती है।
Minimum Execution	यदि condition false है तो body एक बार भी execute नहीं होगी।	Loop body कम से कम एक बार execute जरूर होती है।
Syntax (विन्यास)	while(condition) { ... }	do { ... } while(condition);
Semicolon Use	Condition के बाद semicolon नहीं होता।	Condition के बाद semicolon अनिवार्य होता है।
Execution Flow	Check → Execute → Repeat (if true)	Execute → Check → Repeat (if true)
Use Case	जहाँ loop body condition true होने पर ही चले।	जहाँ loop body कम से कम एक बार चलाना जरूरी हो।

Example	#include <stdio.h>
	int main() {
	int i = 1;
	
	printf("Using while loop:\n");
	while (i <= 5) {
	printf("Number: %d\n", i);
	i++;
	}
	return 0;
	}
	
	Output:
	Using while loop:
	Number: 1
	Number: 2
	Number: 3
	Number: 4
		#include <stdio.h>
		int main() {
		int i = 1;
		
		printf("Using do-while loop:\n");
		do {
		printf("Number: %d\n", i);
		i++;
		} while (i <= 5);
		return 0;
		}
		
		Output:
		Using do-while loop:
		Number: 1
		Number: 2
		Number: 3
		Number: 4

	Number: 5	Number: 5
Output if condition is false	कुछ नहीं (loop body execute नहीं होगी)	एक बार output (loop body एक बार चलेगी)
Real-life Analogy (उदाहरण)	ATM में पहले pin सही हो तभी आगे जाए	Menu-driven programs: menu पहले show होता है फिर पूछता है कि दुबारा दिखाएं?

Nested Loops

जब एक loop के अंदर दूसरा loop लिखा जाता है, तो उसे Nested Loop (नेस्टेड लूप) कहा जाता है। Nested loops का उपयोग तब किया जाता है जब हमें एक ही तरह की प्रक्रिया को multi-level या multi-dimensional रूप में दोहराना हो।

उदाहरण:

- 2D array में data को access करना
- Pattern printing
- Multiplication tables
- Game logic या multi-level menu systems

Concept:

- Nested loop में, एक outer loop होता है और उसके अंदर एक inner loop।
- हर बार जब outer loop एक बार execute होता है**,** तब inner loop पूरी तरह से execute होता है**।**
- उदाहरण के लिए, अगर outer loop 3 बार चलता है और inner loop 4 बार चलता है, तो inner loop कुल 3 × 4 = 12 बार चलेगा।

Types of Nested Loops:

Outer Loop Type	Inner Loop Type
for	for
for	while
while	for
while	while
do-while	for
etc.	etc.

आप किसी भी प्रकार का loop किसी भी प्रकार के loop के अंदर nest कर सकते हैं।

Syntax of Nested for Loops:

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

Inner loop हर बार outer loop के एक चक्र (cycle) पर पूरा चलता है।

Example 1: 3x3 Matrix Print

```
#include <stdio.h>

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

Output:

1 2 3

1 2 3

1 2 3

Explanation:

- Outer loop (i) 3 times चलेगा (rows)
- Inner loop (j) हर बार 3 values print करेगा (columns)

Example 2: Multiplication Table (1 to 3)

```
#include <stdio.h>


int main() {
   int i, j;
   for(i = 1; i <= 3; i++) {
      for(j = 1; j <= 10; j++) {
         printf("%d x %d = %d\t", i, j, i*j);
      }
      printf("\n");
   }
   return 0;
}
```

Output:

1 x 1 = 1 1 x 2 = 2 ... 1 x 10 = 10

2 x 1 = 2 2 x 2 = 4 ... 2 x 10 = 20

3 x 1 = 3 3 x 2 = 6 ... 3 x 10 = 30

Working in Detail:

- Outer loop एक बार चलता है।
- उस एक बार में inner loop पूरी तरह execute होता है।
- Inner loop खत्म होने के बाद outer loop फिर से increment होकर next iteration पर जाता है।
- यह प्रक्रिया तब तक दोहराई जाती है जब तक outer loop की condition false न हो जाए।

Use Cases (उपयोग):

Use Case	Explanation
Pattern Printing	Pyramid, triangle, square patterns
2D Array Traversal	जैसे matrix में row और column से data access करना
Tables and Chart Generation	Multiplication table, tabular data
Nested Menu/Options Systems	Console-based applications में multi-level menu
Complex Calculations	Statistical or financial operations

Important Points (महत्वपूर्ण बातें):

- Nested loops की complexity ज़्यादा हो सकती है (e.g. $O(n^2)$, $O(n^3)$ etc.)
- ज़्यादा nested loops performance पर असर डाल सकते हैं।
- Inner loop हमेशा outer loop की हर iteration पर पूरी तरह चलता है।
- Variable names में confusion न हो, इसके लिए proper naming करें।
- Loop termination conditions सही होना जरूरी है वरना infinite loop बन सकता है।

Example 3: Character Pattern Using Nested Loops

```
#include <stdio.h>


int main() {
   char ch = 'A';
   int i, j;
   for(i = 1; i <= 5; i++) {
      for(j = 1; j <= i; j++) {
         printf("%c ", ch);
      }
      ch++;
      printf("\n");
   }
```

```
    return 0;
}
```

Output:

```
A
B B
C C C
D D D D
E E E E E
```

Break Statement

break statement का उपयोग C प्रोग्रामिंग में किसी loop (जैसे for, while, do-while) या switch statement के अंदर किया जाता है। इसका मुख्य उद्देश्य है— loop या switch के execution को तुरंत रोकना और control को loop के बाद वाली पहली statement पर भेजना।

Working:

- जब प्रोग्राम control किसी loop के अंदर break statement तक पहुँचता है, तो loop तुरंत terminate हो जाता है, चाहे loop की original condition पूरी हुई हो या नहीं।
- इसके बाद प्रोग्राम control loop या switch के बाद के कोड पर चला जाता है।
- इसका उपयोग तब किया जाता है जब हमें पता चल जाए कि loop को और execute करने की जरूरत नहीं है, जैसे कोई खास condition पूरी हो जाना।

Where it is used:

Context	Explanation
Loop Statements	for, while, do-while को तोड़ने के लिए
Switch Statements	किसी particular case को पूरा करने के बाद switch से बाहर निकलने के लिए

Syntax:

```
break;
```

- यह एक standalone statement है, इसके कोई arguments नहीं होते।
- इसे हमेशा semicolon (;) के साथ लिखा जाता है।

Important Points:

- break केवल सबसे नजदीकी (nearest) loop या switch से बाहर निकलता है। अगर nested loops में break लिखा है, तो वह केवल जिस loop के अंदर है उसी को तोड़ेगा, बाहर वाले को नहीं।
- break infinite loops को रोकने के लिए भी इस्तेमाल किया जाता है।
- Loop के अंदर कहीं भी break लिख सकते हैं, पर logic समझ कर ही लगाएं।
- switch के अंदर break डालना ज़रूरी होता है, ताकि control अगले case में न चले जाए।

Example 1: break in for loop

```
#include <stdio.h>


int main() {
   int i;
   for(i = 1; i <= 10; i++) {
      if(i == 5) {
         break;  // जब i की value 5 होगी तो loop रुक जाएगा
      }
      printf("%d ", i);
   }
   return 0;
}
```

Output:

```
1 2 3 4
```

Explanation: जैसे ही i = 5 हुआ, break statement ने loop को रोक दिया। इसलिए 5 और आगे के नंबर print नहीं हुए।

Example 2: break in while loop

```
#include <stdio.h>


int main() {
   int i = 1;
   while(1) {    // Infinite loop
```

```
        printf("%d ", i);
        if(i == 3) {
            break;  // जब i 3 होगा तो loop रुकेगा
        }
        i++;
    }
    return 0;
}
```

Output:

1 2 3

Explanation: यह एक infinite loop था, लेकिन break के कारण loop तीन बार चलकर रुक गया।

Example 3: break in switch case

```
#include <stdio.h>

int main() {
    int day = 3;
    switch(day) {
        case 1:
            printf("Monday");
            break;
        case 2:
            printf("Tuesday");
            break;
        case 3:
            printf("Wednesday");
            break;
        default:
            printf("Invalid day");
```

```
    }
    return 0;
}
```

Output:

Wednesday

Explanation: यहाँ break ने case 3 को पूरा करने के बाद switch से बाहर निकल दिया, जिससे default case execute नहीं हुआ।

Use Cases of break statement:

Use Case	Explanation
Exit loop on specific condition	जब कोई खास condition पूरी हो जाए तो loop रोकना (जैसे value मिलना)
Infinite loops से बाहर निकलना	अनंत लूप को रोकने के लिए (जैसे user input पर रोक)
Switch case में flow control	अलग-अलग cases के बीच control flow को manage करना
Searching problems	जब कोई element मिल जाए तो loop तुरंत रोक देना

Advantages:

- Loop को जल्दी खत्म कर सकता है जब काम पूरा हो जाए।
- Program का flow बेहतर manage होता है।
- अनावश्यक iterations को avoid करता है, जिससे efficiency बढ़ती है।

Disadvantages / Cautions:

- ज़्यादा break statement से code पढ़ना मुश्किल हो सकता है।
- Nested loops में misuse से bugs आ सकते हैं।
- Structured programming में break के बार-बार इस्तेमाल को avoid करना चाहिए।

continue Statement

continue statement का उपयोग loop के अंदर किया जाता है ताकि loop के current iteration को छोड़कर अगली iteration पर control transfer किया जा सके।

मतलब जब continue statement execute होता है, तब उसके नीचे का जो भी code है, वो skip हो जाता है और loop की condition को चेक करके loop फिर से शुरू होता है।

Purpose (उद्देश्य):

- कुछ conditions में जब आप चाहते हैं कि loop की body का शेष हिस्सा skip हो जाए और control वापस loop के शुरू में चला जाए।
- यह break के विपरीत होता है — break loop को पूरी तरह से terminate करता है, जबकि continue सिर्फ current iteration को छोड़ता है।

Where it is Used:

Loop Type	continue applicable?
for loop	हाँ
while loop	हाँ
do-while loop	हाँ

Syntax:

```
continue;
```

- यह एक standalone statement होता है।
- Semicolon (;) के साथ लिखा जाता है।
- इसके कोई arguments नहीं होते।

Working (कैसे काम करता है):

Loop	continue क्या करता है
for	update expression पर jump करता है और फिर condition check करता है
while	condition पर वापस jump करता है
do-while	condition check करने के लिए वापस jump करता है

Important Points:

- continue सिर्फ loop के अंदर valid होता है।
- इसके नीचे का कोड execute नहीं होता जब यह run होता है।
- Nested loops में ये सिर्फ **current inner loop** को affect करता है।

Example 1: continue in for loop

```
#include <stdio.h>

int main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3) {
            continue;  // skip जब i = 3
        }
        printf("%d ", i);
    }
    return 0;
}
```

Output:

1 2 4 5

Explanation: जब i = 3 होता है, तब continue उस iteration को छोड़ देता है और loop आगे बढ़ता है। इसलिए 3 print नहीं हुआ।

Example 2: continue in while loop

```
#include <stdio.h>

int main() {
    int i = 0;
    while(i < 5) {
        i++;
        if (i == 2) {
            continue;  // i = 2 skip होगा
        }
        printf("%d ", i);
    }
    return 0;
}
```

Output:

1 3 4 5

Explanation: जब i = 2 होता है, तो continue के कारण print statement skip हो जाता है।

Example 3: continue in do-while loop

```
#include <stdio.h>

int main() {
    int i = 0;
    do {
        i++;
        if (i == 4) {
            continue;
        }
        printf("%d ", i);
    } while(i < 5);
    return 0;
}
```

Output:

1 2 3 5

Explanation: जब i = 4 होता है, तब continue printf() को skip कर देता है। इसलिए 4 print नहीं होता।

Use Cases (कहाँ उपयोगी होता है):

Use Case	Description
कुछ values को skip करना हो	जैसे odd या even numbers को ignore करना
Unwanted steps को avoid करना	जैसे validation fail होने पर step skip करना
Large loop body में कुछ हिस्से को execute न करना	Efficiency के लिए

Advantages (लाभ):

- Code को simple और readable बनाता है जब आपको कुछ conditions में loop को skip करना हो।
- Unnecessary processing को avoid करता है।
- Multiple condition checks को manage करना आसान हो जाता है।

Disadvantages (हानियाँ / सावधानियाँ):

- बहुत ज्यादा continue statements से logic complex हो सकता है।
- Beginners के लिए समझना थोड़ा confusing हो सकता है।
- यदि carefully नहीं लिखा गया, तो logic errors और infinite loops भी हो सकते हैं।
- Difference between break and continue Statements

Feature	break Statement	continue Statement
Definition	break is used to terminate the loop or switch statement entirely.	continue is used to skip the current iteration of the loop and go to the next one.
Use/Behavior	Ends the loop or switch execution immediately.	Skips remaining part of loop body and continues with the next iteration.
Control Flow	Control jumps out of the loop or switch block.	Control jumps to the beginning of loop to evaluate the condition.
Placement	Can be used inside for, while, do-while, and switch.	Can be used inside for, while, and do-while only.
Typical Use Case	Used when a specific condition forces early exit from the loop or switch.	Used when a specific condition requires skipping the rest of the loop's body.
Effect on Loop Execution	Loop is terminated completely; subsequent iterations don't occur.	Loop is not terminated, only current iteration is skipped.
Syntax	break;	continue;

Example	#include <stdio.h>
	int main() {
	int i;
	
	for(i = 1; i <= 5; i++) {
	if(i == 3) {
	break;
	
	// loop will terminate when i == 3
	}
	printf("Value of i: %d\n", i);
	}
		#include <stdio.h>
		int main() {
		int i;
		
		for(i = 1; i <= 5; i++) {
		if(i == 3) {
		continue;
		
		// skip the rest and go to next iteration
		}
		printf("Value of i: %d\n", i);

	return 0; } Output: Value of i: 1 Value of i: 2	} return 0; } Output: Value of i: 1 Value of i: 2 Value of i: 4 Value of i: 5
Applicable Inside Switch?	Yes	No
Control Transfer To	Statement after the loop or switch block.	Loop condition or next iteration update expression.
Code Execution After Statement	Skipped as control moves outside the loop/switch.	Only the remaining code of the current iteration is skipped.
Can it create infinite loop?	Not usually (since it exits loop)	If misused without proper condition, it may lead to infinite loop

Unit-4
User Defined Function

Function - Function C भाषा में एक ऐसा self-contained, reusable block of code होता है जो किसी विशेष कार्य (specific task) को पूरा करता है।C प्रोग्रामिंग में function एक ऐसा नामित (named) और स्वतंत्र (independent) code block होता है जो किसी विशेष कार्य को पूरा करने के लिए लिखा जाता है। इसे जरूरत पड़ने पर call किया जाता है। Function इनपुट ले सकता है (arguments), प्रक्रिया कर सकता है और परिणाम (return value) दे सकता है।

जब कोई कार्य बार-बार करना हो या उसे logically अलग करना हो, तो हम उसे एक function में define कर देते हैं।

Function को define करने के बाद उसे किसी भी जगह से call किया जा सकता है, जिससे code बार-बार लिखने की ज़रूरत नहीं होती। इससे modularity और reusability बढ़ती है।

Need of function in C:

- **Modularity (मॉड्यूलरिटी):**
  Function की मदद से किसी बड़े program को छोटे-छोटे logical parts में divide किया जा सकता है। इससे program को समझना और manage करना आसान हो जाता है।
- **Reusability (पुनः उपयोग की क्षमता):**
  एक बार function को define कर देने के बाद, हम उसे program में कई बार call कर सकते हैं। इससे code को बार-बार नहीं लिखना पड़ता।
- **Debugging आसान होती है:**
  जब कोई error आती है, तो हम सीधे उसी function को check और correct कर सकते हैं जहाँ problem है। इससे error ढूंढना और fix करना आसान होता है।
- **Readability & Maintainability (पढ़ने और maintain करने में आसान):**
  Functions program को पढ़ने योग्य और साफ-सुथरा बनाते हैं। इससे code का maintenance future में भी आसान होता है।
- **Structured Programming का आधार:**
  C एक structured programming language है, और functions इस structure को बनाए रखने में मुख्य भूमिका निभाते हैं। ये logical flow को define करने में मदद करते हैं।

Types of function in C:

Library Functions (लाइब्रेरी फंक्शन) - Library functions वे functions होते हैं जो पहले से ही C की standard libraries में बने हुए होते हैं। इन functions को C compiler के द्वारा provide किया जाता है और ये अक्सर किसी विशेष task को करने के लिए उपयोग किए जाते हैं, जैसे input लेना, output दिखाना, mathematical calculations करना, string operations करना आदि।

इन functions को इस्तेमाल करने के लिए programmer को केवल proper header file include करनी होती है — जैसे कि stdio.h for printf() और scanf(), math.h for sqrt() और pow(), string.h for strlen() और strcpy() आदि।

इन functions का syntax पहले से defined होता है और इन्हें redefine करने की जरूरत नहीं होती। उदाहरण के लिए, printf() function का उपयोग output स्क्रीन पर text या variable की value को दिखाने के लिए किया जाता है, जबकि scanf() का उपयोग user से input लेने के लिए किया जाता है। ये सभी functions program को आसान और efficient बनाने में मदद करते हैं।

Examples:

printf(), scanf(), sqrt(), strlen(), getch(), puts(), gets(), etc.

User-defined Functions (यूज़र डिफाइंड फंक्शन) - User-defined functions वो functions होते हैं जो programmer खुद define करता है किसी specific task को perform करने के लिए। जब कोई task बार-बार program में use होना हो या किसी logic को अलग block में define करके reusable बनाना हो, तब user-defined function बहुत उपयोगी साबित होते हैं। इन functions को तीन parts में divide किया जाता है — function declaration (prototype), function definition और function call.

1. **Function Declaration (Function Prototype)**

   यह बताता है कि function किस नाम का है, क्या return करेगा और उसमें कौन-कौन से arguments होंगे।

   Syntax:

   return_type function_name(parameter list);

   Example:

   int sum(int a, int b);

2. **Function Definition**

   यह actual block होता है जिसमें function का काम लिखा जाता है।

   Syntax:

   return_type function_name(parameter list) {

   // statements

```
    return value;
}
```

Example:

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

3. **Function Call**

Function को execute करवाने के लिए उसे call किया जाता है।

Syntax:

```
function_name(actual arguments);
```

Example:

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

Types of User-defined Functions in C (Based on Arguments and Return Type): C language में user-defined functions को arguments और return value की presence के आधार पर चार प्रकारों में divide किया जाता है:

1. **Function with No Argument and No Return Value**

विशेषता:

- यह function कोई input (argument) नहीं लेता।
- यह function कोई result return भी नहीं करता।
- इसे सिर्फ कोई fixed task perform करने के लिए बनाया जाता है।

Syntax:

```
void function_name() {
   // statements
}
```

Example:

```
#include <stdio.h>
void greet() {
    printf("Hello, Welcome to C Programming!");
}
int main() {
    greet();
    return 0;
}
```

Output:

```
Hello, Welcome to C Programming!
```

2. Function with Arguments but No Return Value

विशेषता:

- यह function input parameters (arguments) लेता है।
- लेकिन कोई value return नहीं करता।
- Task perform करने के लिए बाहर से values pass करनी होती हैं।

Syntax:

```
void function_name(data_type arg1, data_type arg2) {
   // statements
}
```

Example:

```
#include <stdio.h>
void displaySum(int a, int b) {
```

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

Output:

```
Sum = 30
```

3. Function with No Argument but Return Value

विशेषता:

- यह function कोई argument नहीं लेता।
- लेकिन एक value return करता है।
- सभी calculations function के अंदर होते हैं, और result वापस भेजा जाता है।

Syntax:

```
data_type function_name() {
    // statements
    return value;
}
```

Example:

```
#include <stdio.h>
int getNumber() {
    int num = 100;
    return num;
}
```

```
int main() {
    int result = getNumber();
    printf("Returned value: %d", result);
    return 0;
}
```

Output:

```
Returned value: 100
```

4. Function with Arguments and Return Value

विशेषता:

- यह सबसे flexible और commonly used function type है।
- यह input parameters लेता है और एक value return भी करता है।
- Logic, input और output तीनों handle किए जाते हैं।

Syntax:

```
data_type function_name(data_type arg1, data_type arg2) {
    // statements
    return value;
}
```

Example:

```
#include <stdio.h>
int multiply(int a, int b) {
    return a * b;
}
int main() {
    int result = multiply(5, 6);
```

```
    printf("Multiplication = %d", result);
    return 0;
}
```

Output:

```
Multiplication = 30
```

Math Predefined Functions in C (“<math.h>” Header File)

1. sqrt() – Square Root निकालने के लिए

Syntax:

```
double sqrt(double x);
```

Description: यह function किसी number का square root निकालता है।

Example:

```
#include <stdio.h>
#include <math.h>
int main() {
   double result = sqrt(25.0);
   printf("Square root of 25 = %.2f", result);
   return 0;
}
```

Output:

```
Square root of 25 = 5.00
```

2. pow() – Power Function

Syntax:

```
double pow(double base, double exponent);
```

Description: base को exponent की power तक raise करता है (base^exponent).

Example:

```
#include <stdio.h>
#include <math.h>
int main() {
   double result = pow(2.0, 3.0);
   printf("2 raised to power 3 = %.2f", result);
   return 0;
}
```

Output:

```
2 raised to power 3 = 8.00
```

3. fabs() – Absolute Value (Positive value return करता है)

Syntax:

```
double fabs(double x);
```

Description: यह किसी number का absolute (positive) value return करता है।

Example:

```
#include <stdio.h>
#include <math.h>
int main() {
   double result = fabs(-9.7);
   printf("Absolute value = %.2f", result);
   return 0;
}
```

Output:

```
Absolute value = 9.70
```

4. ceil() – Ceiling Function

Syntax:

```
double ceil(double x);
```

Description: यह function किसी number को **upper integer** (निकटतम बड़ा पूर्णांक) में round करता है।

Example:

```
#include <stdio.h>
#include <math.h>
int main() {
   double result = ceil(4.3);
   printf("Ceiling of 4.3 = %.2f", result);
   return 0;
}
```

Output:

```
Ceiling of 4.3 = 5.00
```

5. floor() – Floor Function

Syntax:

```
double floor(double x);
```

Description: यह function किसी number को **lower integer** (निकटतम छोटा पूर्णांक) में round करता है।

Example:

```
#include <stdio.h>
#include <math.h>
int main() {
   double result = floor(4.7);
```

```
    printf("Floor of 4.7 = %.2f", result);
    return 0;
}
```

Output:

```
Floor of 4.7 = 4.00
```

6. fmod() – Floating Point Modulus

Syntax:

```
double fmod(double x, double y);
```

Description: यह दो floating point numbers का modulus देता है (remainder).

Example:

```
#include <stdio.h>
#include <math.h>
int main() {
    double result = fmod(10.5, 3.0);
    printf("Remainder = %.2f", result);
    return 0;
}
```

Output:

```
Remainder = 1.50
```

7. log() – Natural Logarithm (Base e)

Syntax:

```
double log(double x);
```

Description: यह function natural logarithm निकालता है (base **e** का log).

Example:

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

```
int main() {
    double result = log(2.7183); // approximately e
    printf("Log = %.2f", result);
    return 0;
}
```

Output:

```
Log = 1.00
```

8. log10() – Logarithm base 10

Syntax:

```
double log10(double x);
```

Description: यह function base-10 का log निकालता है।

Example:

```
#include <stdio.h>
#include <math.h>
int main() {
    double result = log10(1000);
    printf("Log base 10 = %.2f", result);
    return 0;
}
```

Output:

```
Log base 10 = 3.00
```

9. sin(), **cos()**, **tan() –** Trigonometric Functions

Syntax:

```
double sin(double x);
double cos(double x);
double tan(double x);
```

Description: यह angles (radians में) के लिए trigonometric values return करते हैं।

Example:

```
#include <stdio.h>
#include <math.h>
int main() {
    double angle = 3.14159 / 2; // 90 degrees in radians
    printf("Sin(90°) = %.2f\n", sin(angle));
    printf("Cos(90°) = %.2f\n", cos(angle));
    return 0;
}
```

Output:

```
Sin(90°) = 1.00
Cos(90°) = 0.00
```

String Functions in C ("<string.h>" **Header File)**

1. strlen() – String की Length निकालना

Syntax:

```
int strlen(const char *str);
```

Description: यह function किसी string की length (characters की संख्या) return करता है, लेकिन null character (**'\0'**) को count नहीं करता।

Example:

```
#include <stdio.h>
#include <string.h>
int main() {
    char name[] = "India";
    int length = strlen(name);
```

```
    printf("Length of string = %d", length);
    return 0;
}
```

Output:

```
Length of string = 5
```

2. strcpy() – एक string को दूसरी string में copy करना

Syntax:

```
char *strcpy(char *dest, const char *src);
```

Description: यह source string (src) को destination string (dest) में copy करता है।

Example:

```
#include <stdio.h>
#include <string.h>
int main() {
    char str1[] = "Hello";
    char str2[20];
    strcpy(str2, str1);
    printf("Copied string = %s", str2);
    return 0;
}
```

Output:

```
Copied string = Hello
```

3. strcat() – दो strings को जोड़ना

Syntax:

```
char *strcat(char *dest, const char *src);
```

Description: यह destination string के अंत में source string को जोड़ देता है।

Example:

```
#include <stdio.h>
#include <string.h>
int main() {
   char str1[30] = "Good ";
   char str2[] = "Morning";
   strcat(str1, str2);
   printf("Concatenated string = %s", str1);
   return 0;
}
```

Output:

```
Concatenated string = Good Morning
```

4. strcmp() – दो strings को compare करना

Syntax:

```
int strcmp(const char *str1, const char *str2);
```

Description: यह दो strings को lexicographically compare करता है:

अगर दोनों बराबर हैं → return 0

अगर str1 > str2 → positive value

अगर str1 < str2 → negative value

Example:

```
#include <stdio.h>
#include <string.h>
int main() {
   char str1[] = "apple";
   char str2[] = "banana";
   int result = strcmp(str1, str2);
   printf("Comparison result = %d", result);
   return 0;
```

```
}
```

Output:

```
Comparison result = -1 (or any negative value)
```

5. strrev() – String को reverse करना

Syntax:

```
char *strrev(char *str);
```

Description: यह string को उल्टा कर देता है।

Example:

```
#include <stdio.h>
#include <string.h>
int main() {
   char str[] = "Hello";
   strrev(str);
   printf("Reversed string = %s", str);
   return 0;
}
```

Output:

```
Reversed string = olleH
```

6. strlwr() – String को lowercase में convert करना

Syntax:

```
char *strlwr(char *str);
```

Example:

```
#include <stdio.h>
#include <string.h>
int main() {
```

```
    char str[] = "HELLO";
    strlwr(str);
    printf("Lowercase = %s", str);
    return 0;
}
```

Output:

```
Lowercase = hello
```

7. strupr() – String को uppercase में convert करना

Syntax:

```
char *strupr(char *str);
```

Example:

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[] = "world";
    strupr(str);
    printf("Uppercase = %s", str);
    return 0;
}
```

Output:

```
Uppercase = WORLD
```

User-Defined Function in C: C Language में User-Defined Function एक ऐसा function होता है जिसे programmer खुद define करता है किसी specific task को perform करने के लिए। यह function, C के built-in (जैसे printf(), scanf() आदि) functions की तरह ही होता है, लेकिन इसे हम अपनी ज़रूरत के अनुसार खुद बनाते हैं।

User-Defined Function वह function होता है जिसे programmer खुद define करता है किसी विशेष कार्य को बार-बार करने के लिए।

Structure of a User-Defined Function (Function की संरचना):

Function Declaration (Prototyping)

Function Definition

Function Call

Example:

```
// 1. Declaration
int add(int, int);
int main() {
   // 3. Calling
   int result = add(5, 10);
   printf("Result = %d", result);
   return 0;
}
// 2. Definition
int add(int a, int b) {
   return a + b;
}
```

Need of User-Defined Function in C:

1. **Reusability (पुनः उपयोग करना):** एक बार function define कर देने के बाद, आप उसे program में कई बार call कर सकते हैं। इससे code बार-बार नहीं लिखना पड़ता।
2. **Modularity (कार्य को भागों में बाँटना):** Program को छोटे-छोटे हिस्सों में divide करने से उसे समझना और manage करना आसान हो जाता है।
3. **Easy Debugging (त्रुटि निकालना आसान):** अगर किसी specific task में error हो, तो सिर्फ उसी function को debug किया जा सकता है।
4. **Improved Readability (पढ़ने में आसानी):** Functions के use से program को logically पढ़ना और समझना सरल हो जाता है।
5. **Avoid Redundancy (बार-बार कोड न लिखना):** बार-बार एक जैसा काम करने वाले code को function में डालकर redundancy हटाई जाती है।
6. **Structured Programming का मुख्य हिस्सा:** C structured programming language है, और functions इसकी backbone होते हैं।

Unit-5
Single Dimensional Array in 'C'

Array in C - An array is a collection of elements of the same data type stored in contiguous memory locations.
Array एक ऐसा डेटा स्ट्रक्चर है जिसमें एक ही प्रकार (same data type) के कई values को एक साथ एक ही नाम से store किया जाता है।

Need of array -

- जब हमें एक ही प्रकार के **multiple values** को manage करना हो जैसे कि 100 students के marks, तो हर एक के लिए अलग-अलग variable बनाना मुश्किल हो जाता है।
- Array से हम एक ही नाम से सभी values को index की मदद से access कर सकते हैं।

Syntax:

```
data_type array_name[size];
```

Example:

```
int marks[5];
```

इसका मतलब है कि marks नाम का एक array है जो 5 integers को store कर सकता है।

Types of Arrays:

- One-Dimensional Array
- Two-Dimensional Array
- Multi-Dimensional Array

Advantages of Array (Array के लाभ):

- एक ही नाम से multiple values को handle करना
- Memory और time saving
- Loop के द्वारा आसानी से traverse कर सकते हैं

Limitations of Array (सीमाएँ):

- Fixed size होता है (size को runtime में change नहीं कर सकते)
- केवल एक ही data type के values store कर सकते हैं

Declaration of one-dimensional array –

Syntax –

```
data_type array_name[size];
```

Example –

```
int numbers[5];      // 5 integers का array
float marks[10];     // 10 float values का array
char vowels[5];      // 5 characters का array
```

- int, float, char = data types हैं
- numbers, marks, vowels = array के नाम हैं
- [5], [10] = array की size है (यानि कितने elements store होंगे)

Initialization of array –

(a) सभी Values एक साथ देना:

```
int numbers[5] = {10, 20, 30, 40, 50};
```

यह array में सभी values सेट कर देता है।

(b) Partial Initialization:

```
int numbers[5] = {10, 20};
```

बाकी values को **default 0** मान लिया जाता है।

(c) Size के बिना Initialization:

```
int numbers[] = {10, 20, 30};
```

Compiler अपने आप size को count कर लेता है (यहाँ size = 3)

Access to elements of array – Array के किसी भी element को उसकी index से access किया जाता है:

```
numbers[0] = 25;          // पहला element बदलना
printf("%d", numbers[1]);  // दूसरा element print करना
```

Note: Index 0 से शुरू होता है, यानि numbers[0] पहला element है।

Example Program –

```
#include <stdio.h>
int main() {
   int i;
   int numbers[5] = {1, 2, 3, 4, 5};
   for(i = 0; i < 5; i++) {
      printf("%d ", numbers[i]);
   }
   return 0;
}
```

Output:

1 2 3 4 5

Array Operations –

1. **Insertion in Array (Array में डाटा जोड़ना) -** Array में किसी specific position पर नया element insert करना होता है।

   Steps:

   - Array की size check करें कि जगह है या नहीं।
   - सभी elements को उस position से एक step पीछे shift करें।
   - नए element को insert करें।

   Example:

```
#include <stdio.h>
int main() {
    int arr[6] = {10, 20, 30, 40, 50};
    int i, pos = 2, newVal = 25;
    int size = 5;
    for(i = size; i > pos; i--) {
        arr[i] = arr[i-1];
    }
    arr[pos] = newVal;
    size++;
    for(i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output:

10 20 25 30 40 50

2. **Searching in Array (Array में Value खोजना)-** किसी दिए गए value को array में find करना होता है।

Linear Search:

```
#include <stdio.h>
int main() {
    int arr[] = {5, 10, 15, 20, 25};
    int key = 15, found = 0, i;
    for(i = 0; i < 5; i++) {
        if(arr[i] == key) {
            printf("Element found at index %d\n", i);
            found = 1;
            break;
        }
    }
    if(!found)
        printf("Element not found\n");
    return 0;
}
```

Output:

Element found at index 2

3. **Deletion in Array (Array से Element हटाना) -** किसी position या value को हटाकर बाकी elements को shift किया जाता है।

Example:

```
#include <stdio.h>
int main() {
    int arr[5] = {10, 20, 30, 40, 50};
    int pos = 2, i, size = 5;
    for(i = pos; i < size - 1; i++) {
        arr[i] = arr[i + 1];
    }
    size--;
    for(i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output:

```
10 20 40 50
```

4. **Concatenation of Two Strings (दो Strings को जोड़ना) -** दो strings को एक साथ जोड़ने के लिए strcat() function use होता है।

Example:

```
#include <stdio.h>
#include <string.h>
int main() {
    char str1[50] = "Hello ";
    char str2[] = "World";
    strcat(str1, str2);
    printf("Concatenated String: %s", str1);
    return 0;
```

```
}
```

Output:

```
Concatenated String: Hello World
```

- strcat() के लिए destination string (str1) में पर्याप्त जगह होना जरूरी है।