Data Representation in C Programming

1. Introduction

Data representation means the way data is stored and interpreted in a computer. In C programming, different types of data (like integers, floating point numbers, characters, etc.) are stored in memory using specific formats and rules. The same data can have different representations in memory and on screen.

2. Types of Data Representation in C

1. Integer Representation
2. Floating Point Representation
3. Character Representation
4. Boolean Representation
5. String Representation (array of characters)

3. Integer Representation

- Integers are stored in binary form (0 and 1).
- C supports different sizes: short, int, long, long long.
- The number system used is usually signed (two's complement) for negative numbers.
- Example:

Type	Size (bits)	Range (approx.)
short int	16	-32,768 to 32,767
int	32	-2,147,483,648 to 2,147,483,647
long int	32/64	(platform dependent)
long long int	64	-9 quintillion to 9 quintillion

Example in C:

```
int a = 10;   // stored as 00000000 00000000 00000000 00001010
              (in 32-bit system)
```

4. Floating Point Representation

- Floating point numbers are stored in IEEE 754 format.
- It uses 1 sign bit, 8 exponent bits and 23 mantissa bits (for float).
- Example: 3.14 is stored as a binary representation with sign, exponent and mantissa.

```
float 3.14 = 0  10000001  10010001111010111000011
             ↓     ↓              ↓
            sign  exponent      mantissa
```

5. Character Representation

- Characters are stored using ASCII (American Standard Code for Information Interchange).
- Each character is represented by a numeric value (usually 1 byte = 8 bits).
- Example:

Character	ASCII Value	Binary (8 bits)
A	65	01000001
a	97	01100001
0	48	00110000
\n	10	00001010

6. Boolean Representation

- C does not have a separate boolean type (in older versions).
- It uses int type where 0 represents false and non-zero represents true.
- Example:

```
int flag = 1;     // true
int flag = 0;     // false
```

7. String Representation

- Strings are stored as an array of characters.
- Each character is stored in 1 byte (ASCII).
- The string ends with a null character '\0' (value 0).
- Example:

```
char name[] = "Kartik";
// Stored as: 01001011 01100001 01110010 01110100
              01101011 01101000 00000000
// '\0' at the end
```

8. Conclusion

Data representation is the base of C programming. It helps the computer to store, process and display data correctly. Understanding how different types of data are represented in memory is important for writing efficient and error-free programs.

Problem Analysis in C Programming

1. What is Problem Analysis ?

Problem analysis is the first and most important step in solving a problem using a computer program. It means understanding the problem properly, identifying the input, output, required data, and the logic needed to solve it before actually writing the C program.

2. Why is it Important ?

- Helps to understand the problem clearly.
- Avoids mistakes during coding.
- Saves time and effort.
- Makes the program more efficient and correct.

Key Points to Analyze

- Input
- Output
- Data type
- Variables
- Logic / Algorithm
- Constraints (if any)

3. Steps in Problem Analysis

① Understand the problem → Read the problem statement carefully and find what is given and what is required.

② Identify the Input → Find what data is needed from the user or from a file.

③ Identify the Output → Decide what result should be displayed or produced.

④ Analyze the Logic → Think about the steps and method to solve the problem (algorithm).

⑤ Design the Program → Plan the solution using flowchart or pseudocode.

⑥ Write the Code → Convert the logic into C language.

⑦ Test and Debug → Check if the program gives correct output.

4. Example

Suppose we need to write a program to find the sum of two numbers in C.

- Input = Two numbers (a, b)
- Output = Sum (a + b)
- Logic = Read a, b → add them → display the result.

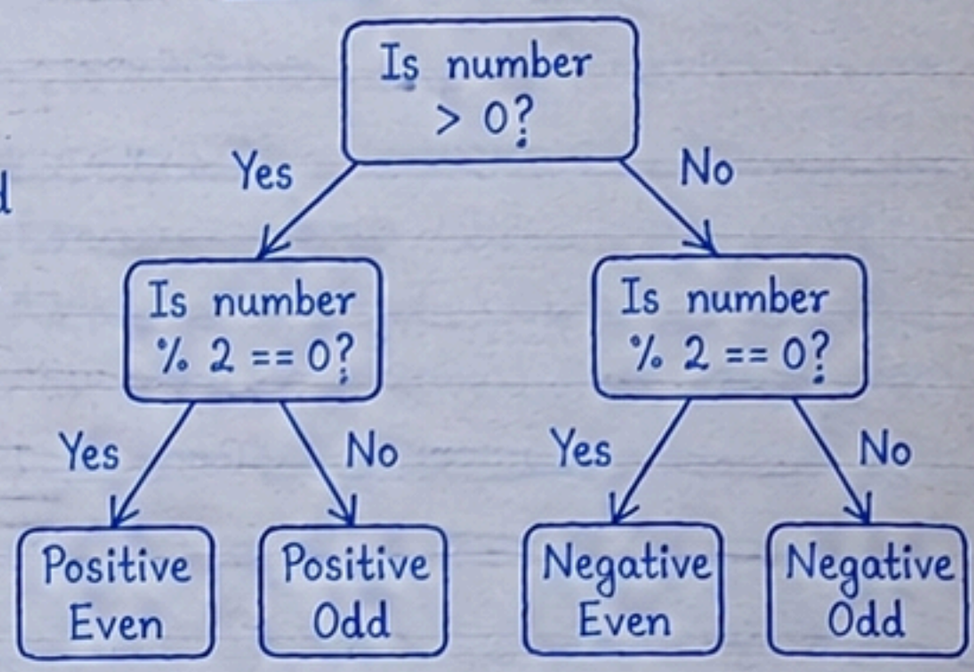
This is the problem analysis. After this, we can draw a flowchart or write the C code to solve it.

Decision Trees / Tables in C Programming

1. What is Decision Tree?

A Decision Tree is a graphical representation of a problem that shows the different conditions and possible outcomes. It helps to make decisions based on the given conditions. In C programming, we use if-else statements (or switch case) to implement the logic shown in a decision tree.

Example Decision Tree



2. What is Decision Table?

A Decision Table is a tabular form that shows all possible conditions and the corresponding actions or results. It is useful when there are multiple conditions and combinations. In C programming, we can use if-else if-else or switch case to implement the logic from a decision table.

Example Decision Table

Condition 1 (Num > 0)	Condition 2 (Num % 2 == 0)	Result
Yes	Yes	Positive Even
Yes	No	Positive Odd
No	Yes	Negative Even
No	No	Negative Odd

3. Decision Tree in C Programming

In C, we use if-else statements to represent a decision tree. Each condition is checked in sequence, and the program follows the path based on the result.

Example:

Write a program to check whether a number is positive, negative or zero.

```
#include <stdio.h>
int main() {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);

    if (n > 0)
        printf("Positive\n");
    else if (n < 0)
        printf("Negative\n");
    else
        printf("Zero\n");

    return 0;
}
```

4. Decision Table in C Programming

A decision table can be implemented using if-else if-else or switch-case statements. It is useful when there are multiple conditions.

Example:

Write a program to find the largest of two numbers using a decision table.

```
#include <stdio.h>
int main() {
    int a, b;
    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);
    if (a > b)
        printf("%d is largest\n", a);
    else
        printf("%d is largest\n", b);
    return 0;
}
```

5. Conclusion

Decision trees and decision tables help us to organize and simplify the logic of a program. They make the code easier to understand, write and debug in C programming.

Pseudo Code and Algorithms in C Programming

1. What is an Algorithm?

An algorithm is a step-by-step set of instructions used to solve a problem or perform a specific task. It tells us what to do, in what order and how to do it. An algorithm is independent of any programming language and can be written in simple English or in a structured format.

Key points about Algorithm:

- It is finite (has a last step).
- It is precise and unambiguous.
- It takes some input and produces output.
- It is language independent.
- It is used to create a program (in C or any language).

Example of an Algorithm

(Find the sum of two numbers)

1. Start
2. Read the first number (a)
3. Read the second number (b)
4. sum = a + b
5. Display the sum
6. Stop

2. What is Pseudo Code?

Pseudo code is a simple, natural language-like representation of an algorithm. It uses English words and simple programming-like structures (such as READ, PRINT, IF, ELSE, FOR, etc.). It is not actual code, but it is very close to C programming and helps in designing the program before writing the real code.

Key points about Pseudo Code:

- It is written in simple English with programming terms.
- It is easy to understand and modify.
- It is not language specific.
- It helps in converting the logic into actual C code.

Pseudo Code Example

(Find the largest of two numbers)

```
1. Start
2. Read a, b
3. If a > b then
       print a
   else
       print b
4. Stop
```

3. Algorithm and Pseudo Code in C Programming

First, we write the algorithm or pseudo code to understand the logic. Then, we convert it into a C program.

Example: Find the sum of two numbers

Algorithm

```
1. Start
2. Read a, b
3. sum = a + b
4. Display sum
5. Stop
```

C Program

```c
#include <stdio.h>
int main()
{
    int a, b, sum;
    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);
    sum = a + b;
    printf("Sum = %d", sum);
    return 0;
}
```

4. Conclusion

Pseudo code and algorithms help us plan the solution, understand the logic and write the C program easily. They make the programming process simple, organized and error-free.

Fundamentals Character Set in C Programming

1. What is Character Set ?

The character set in C programming is the collection of all valid characters that can be used to write programs. These characters include letters, digits, special symbols and some control characters.

Examples of Characters

Letters : A, b, Z, m
Digits : 0, 1, 5, 9
Symbols : +, -, *, /, =
White Space : space, tab, \n
Control : \n, \t, \0

2. Types of Characters in C

The character set in C is divided into the following types:

(a) Alphabets (Letters)

- Both uppercase and lowercase letters are allowed.
- A – Z and a – z

(b) Digits

- The digits 0 to 9 are used for numbers.
- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

(c) Special Symbols

- These are used for operators, delimiters and punctuation.
- Examples: + - * / % = < > ! ? ; , . ( ) { } [ ]

(d) White Space Characters

- Used to separate tokens (words).
- Examples: space, tab, newline, carriage return, form feed.

(e) Control Characters

- Used to control the behavior of the program or terminal.
- Examples: \n (newline), \t (tab), \r (carriage return), \0 (null).

3. Table of Character Set in C

Type	Characters	Example
Uppercase Letters	A – Z	A, B, C, ..., Z
Lowercase Letters	a – z	a, b, c, ..., z
Digits	0 – 9	0, 1, 2, ..., 9
Special Symbols	+ - * / = < > ! ? ; , . ( ) { } [ ]	+, -, *, /, ...
White Space	space, tab, newline, ...	\n, \t, space
Control Characters	\n, \t, \r, \0, ...	\n, \t, \0

4. Important Notes

- The character set in C is case sensitive (A and a are different).
- The underscoce (_) is also a valid character (used in identifiers).
- The character set is defined by the C standard (ASCII for most compilers).
- We can use characters in strings and character variables.

5. Conclusion

The character set is the basic building block of C programming. Understanding it helps us write correct and error-free programs.

Identifiers and Key Words in C Programming

1. What are Identifiers ?

Identifiers are the names given to various program elements such as variables, functions, arrays, structures, etc. In C programming, we use identifiers to identify and distinguish different entities in a program.

2. Rules for Identifiers

- An identifier can contain letters (A-Z, a-z), digits (0-9) and underscore (_).
- The first character must be a letter or underscore (_).
- It cannot contain spaces or special symbols (like +, -, *, /, etc.).
- It is case sensitive (name and Name are different).
- It cannot be a keyword.

3. Examples of Valid and Invalid Identifiers

Valid Identifiers	Invalid Identifiers
a	1num (starts with digit)
sum	sum-1 (contains -)
total1	totat (space)
_count	@name (special symbol)
studentName	int (keyword)
age2	

Example :

```
int age;
float total_marks;
char _name;
double value1;
```

Here, age, total_marks, _name and value1 are valid identifiers.

2. What are Key Words ?

Key words are reserved words in C language. They have a special meaning and are already defined in the language. We cannot use them as identifiers (variable names, function names, etc.).

3. List of C Key Words

auto	default	float	register	struct	volatile
break	do	for	return	switch	while
case	double	goto	short	typedef	
char	else	if	signed	union	
const	enum	int	sizeof	unsigned	
continue	extern	long	static	void	

4. Example

The following are invalid because they are keywords :

```
int = 10;     // invalid (int is a keyword)
return = 5;   // invalid (return is a keyword).
```

5. Conclusion

Identifiers are user-defined names for program elements, while key words are reserved words with special meaning in C. We must follow the rules for identifiers and avoid using key words as names.

Data Types in C Programming

Data types in C define the type of data a variable can store.
C provides several built-in data types, which are mainly divided into the following categories:

1. Primary (Basic) Data Types

Data Type	Size (typically)	Range (approximantely)	Example	Description
char	1 byte	−128 to 127	char ch = 'A';	Stores a single character.
short int	2 bytes	−32,768 to 32,767	short int s = 10;	Stores a small integer.
int	4 bytes	−2,147,483,648 to 2,147,483,647	int i = 100;	Stores a normal integer.
long int	4 or 8 bytes	−2,147,483,648 to 2,147,483,647 (4 bytes) or larger (8 bytes)	long int L = 1000;	Stores a larger integer than int.
float	4 bytes	$\sim 3.4 \times 10^{-38}$ to $\sim 3.4 \times 10^{38}$	float f = 3.14;	Stores a single-precision floating point number.
double	8 bytes	$\sim 1.7 \times 10^{-308}$ to $\sim 1.7 \times 10^{308}$	double d = 3.141592;	Stores a double-precision floating point number.
long double	10 or 16 bytes	$\sim 3.4 \times 10^{-4932}$ to $\sim 1.1 \times 10^{4932}$	long double ld = 3.14L;	Stores a very large range floating point number.
void	–	–	void *p;	Represents no value.

2. Derived Data Types

These are created using primary data types:

- Array
- Pointer
- Structure
- Union
- Function (return type)

3. Enumeration Type

Used to define a set of named integer constants.
e.g. enum day {SUN, MON, TUE};

Constants in C Programming

What is a Constant ?

A constant is a fixed value that does not change during the execution of a program. It is used to represent a value that remains constant throughout the program.

Types of Constants in C

1. **Integer Constants**
   Represent whole numbers (positive or negative) without any decimal part.
   Example : 10, -25, 0, 1000
2. **Floating-Point Constants**
   Represent real numbers (with decimal part).
   Example : 3.14, -0.5, 6.022e23
3. **Character Constants**
   Represent a single character enclosed in single quotes.
   Example : 'A', 'z', '5', '\n'
4. **String Constants**
   Represent a sequence of characters enclosed in double quotes.
   Example : "Hello", "C Programming"
5. **Enumeration Constants**
   Represent named integer values using the enum keyword.
   Example :

```
enum Day { SUN, MON, TUE, WED, THU, FRI, SAT };
enum Day d = MON;    // d = 1
```

Example Program

```
#include <stdio.h>
int main() {
    const int MAX = 100;          // integer constant
    const float PI = 3.14;        // floating-point constant
    const char grade = 'A';       // character constant
    const char *msg = "Hello";    // string constant

    printf("MAX = %d\n", MAX);
    printf("PI = %f\n", PI);
    printf("Grade = %c\n", grade);
    printf("Message = %s\n", msg);
    return 0;
}
```

Key Points :

- Constants are fixed values.
- They improve code readability and maintainability.
- In C, constants can be of different types: integer, float, character, string and enumeration.
- The keyword const is used to declare a constant variable.

Variables and Expressions in C Programming

1. Variables in C Programming

A variable is a named memory location used to store data. The value stored in a variable can be changed during the execution of a program.

Key Points :

- Every variable has a name (identifier).
- It has a data type, which decides what kind of value it can store.
- The value can change, so it is called a variable.
- In C, variables must be declared before they are used.

Syntax :

```
data_type variable_name;
```

Example :

```
int age;            // variable of type int
float price;        // variable of type float
char ch;            // variable of type char
```

Rules for naming a variable :

- It must start with a letter or underscore (_).
- It can contain letters, digits and underscores.
- It cannot contain spaces or special symbols (like @, #, $, etc.).
- It is case sensitive (age and Age are different).

2. Expressions in C Programming

An expression is a combination of variables, constants and operators that produces a value. It is evaluated (computed) by the compiler or at runtime.

Types of Expressions :

1. Arithmetic Expression - e.g. `a + b`, `x - y`, `a * b`
2. Relational Expression - e.g. `a > b`, `x == y`, `m != n`
3. Logical Expression - e.g. `a && b`, `x || y`, `!p`
4. Assignment Expression - e.g. `x = 10`, `y += 5`
5. Bitwise Expression - e.g. `a & b`, `x | y`, `~z`

Example :

```
int a = 5, b = 3;
int sum = a + b;              // arithmetic expression
int isGreater = (a > b);      // relational expression
int result = (a && b);        // logical expression
a = a + 10;                   // assignment expression
```

Summary :

- Variables store data and have a data type.
- Expressions combine values using operators to produce a result.
- Both variables and expressions are essential in writing C programs.

Statements in C programming :

A statement is s ct a single line (or group of lines) of code that tells the computer to perform a particular action. In C, every statement ends with a semicolon (;).

Types of Statements :

1. **Expression Statement** :
   - It evaluates an expression. Usually used for calculations or assigning values.

   Example : `a = b + 5;`

2. **Compound Statement (Block)** :
   - A group of statements enclosed within curly braces { }. It is used in decision making, loops, and functions.

   Example :
   ```
   {
       printf ("Hello");
       printf ("C Programming");
   }
   ```

3. **Selection Statement** :
   - Used to choose between different paths (if, if-else, switch).

   Example :
   ```
   if (a > b) {
       printf ("A is greater");
   }
   ```

4. **Iteration Statement** :
   - Used to repeat a block of statements (for, while, do-while).

   Example :
   ```
   for (i = 0; i < 5; i++){
       printf ("%d", i);
   }
   ```

5. **Jump Statement** :
   - Used to transfer control to another part of the program (break, continue, goto, return).

   Example : `break;`

Some important points :

- Each statement is written on a new line or can be written in a single line.
- A statement may be simple (like an expression) or complex (like a compound statement).
- The semicolon (;) at the end is mandatory (except for block statements).

Symbolic Constants in C programming :

A symbolic constant is a name given to a fixed value. It makes the program more readable and easy to maintain. In C, symbolic constants are created using the #define preprocessor directive.

Syntax : `#define CONSTANT_NAME value`

Example :

```
#define PI   3.14159
#define MAX  100
#define MSG  "Hello"
```

Using symbolic constants :

```
float area;
area = PI * r * r;
printf ("Max value = %d", MAX);
printf ("%s", MSG);
```

Advantages of using symbolic constants :

- Improves code readability.
- Makes the program easier to maintain (change the value in one place).
- Reduces the chance of errors (no need to remember the actual value).
- Useful for defining fixed values like mathematical constants, limits, etc.