Arrays

- Collection of similar datatype.
- its a user define datatype
- in normal datatype we can store single value bt what if we want t store multiple values in a single variable.
- int a
  float f
  char ch
- suppose we wantto store marks of 6 subject.
- all elements in array share the same name
- bt when we write int [5]; In this case we can store 5 values in a single variable.
- Index always starts from 0
- also define char or float array
- array is a collection of similar data type we cant add values {1,a,45.3 } like this
- suppose we have to store roll no of students so can we take 10 different variables imagine if there are 100 students how hectic that will be and how memory will be managed
- solution to this we use arrays.
- store different money in different placce

Declaration

- Syntax :
  Declaration
  datatype name_of_array [Size]
  eg: int arr [5]
  char ch [10]
  [] is called as subscript operator

Initialization

- Three ways :

1. int arr[] = {1,2,3,4};
2. int arr[5] = {1,2,3,4,5};
3. int arr[5];
   for(i=0; i<5; i++)
   {
   scanf ("%d", &arr[i]);
   }
   For output again loop
   for(i=0; i<5; i++)
   {

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

}

for single variable -> printf("\n %d " , arr[i]);

2D- Arrays

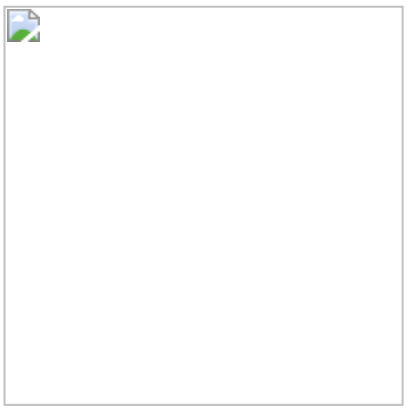
- The array with 2 dimensions can be called as 2-D array or matrix.
- array of 2 x 3 have 2 rows and 3 columns
- syntax

  datatype varible_name [rows] [column];
- eg :

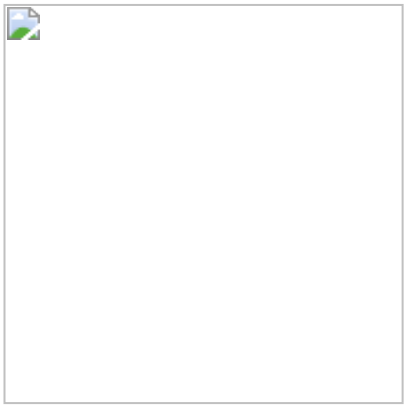
  int arr[2][3] = {1,2,3,4,5,6}.
- for input and output we have use nested for loop.

Process of Compilation

compilation is the process of translating the high level program code ito low level machine code. The compilation process is like taking raw materials and turning them into a finished product, ready to be used.



Four Stages of Compilation



1. Preprocessing

- The first stage is called Preprocessing. The preprocessor looks for special instructions in your code, like macros and #include directives, and expands them before the actual compilation starts.
- The preprocessor handles directives like #include <stdio.h> and replaces macro definitions with their corresponding values. It also removes comments from the code.
- Think of this stage as preparing ingredients before cooking a meal. You gather all the necessary ingredients, like vegetables and spices, and chop or mix them before you start cooking. Similarly, the preprocessor prepares your code by including header files and macros so that it’s ready for the next stage.
- In this stage, comments and extra spaces are also removed, making the code cleaner.
- Once preprocessing is complete, we move on to the next stage.

2. Compilation

- After preprocessing, we enter the Compilation stage. Here, the preprocessed code is converted into assembly language, which is a low-level representation that’s easier for the machine to understand.
- In this stage, the compiler takes your preprocessed code and translates it into assembly language. Assembly language is more readable for the machine but still requires further translation.
- Imagine you're designing a house. The compilation stage is like converting your raw ideas and sketches into a blueprint. The blueprint is not the final house, but it provides a clear structure of how things will be built. Similarly, assembly language provides a structure for your code.
- This is where the compiler checks for syntax errors and issues like missing semicolons, incorrect variable declarations, etc. If your code has no errors, the compiler moves on to the next step.

3. Assembly

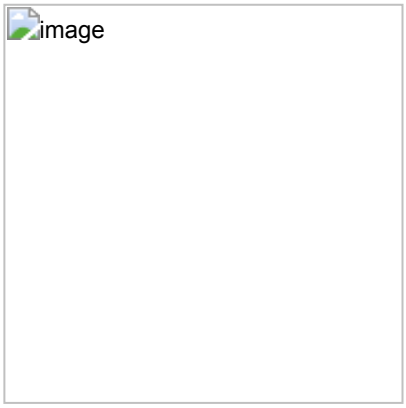
- The third stage is the Assembly stage. Here, the assembly code generated in the previous step is converted into machine code—the language that the computer understands directly.
- The assembler converts the assembly language into machine code (binary), which consists of 0s and 1s. This code is stored in an object file with a .o or .obj extension.
- Let’s go back to our house analogy. The assembly stage is like taking the blueprint and turning it into a digital model. The model represents the house accurately, but it’s still not ready to live in. Similarly, machine code is almost ready to be executed but needs some final touches.
- The machine code in the object file can be understood by the machine, but it’s not yet ready to run as a full program. That happens in the final stage.

4. Linking

- The final stage is Linking. During this stage, the linker combines your object files with any necessary external libraries and creates the final executable file, which you can run on your system.
- In the linking stage, the linker takes all the object files and links them together. It also resolves external references to functions or variables defined in different files or libraries, such as printf() from the C standard library.
- Think of this stage as the final construction of your house. You’ve got the materials, the digital model, and the structure in place, but now you need to bring in plumbing, electricity, and furniture to make it livable. Linking adds these external 'plumbing' and 'furniture' components, creating a fully functional executable.
- Once the linker resolves all references and includes all required libraries, it produces an executable file that you can run on your computer. This executable file has a .exe extension on Windows and an executable format on Linux.

Control Structures

- Till now we have work on program which executes line by line, one after another.
- but now we want to control the sequence of program.
- eg : after the second line i want to execute 4th line.
- int main()
  {
  printf("Hello");
  printf("Fortune");
  printf("cloud");
  }
- In this code there is a fixed sequence means first hello then forune and then cloud will print.
- but I want to display fortune cloud hello.
- In that case we use control statements.
- There are Three types of control statements.



Conditional/ Decision Control Statements

- Decides which instruction will execute next based on some conditions.
- From ours 10 or 12th marks we decide which field to choose.
- means based on some conditions based on marks we decides what to choose means take action.
- when we can use decision control statements in real life.
- Ex :
  - if age is 18 or grater than 18 then you are a valid user

    - based on our marks we decide our grades.
    - in clothes store there are multiple option we decides one of them.

- Decision Control Statements are :

    - If
    - If -else
    - If - Else If - Else ladder
    - Nested If - Else
    - Switch case

If

- The if statement is a powerful decision making statement and is used to control the flow of the execution of statements.
- If condition is true then the block of code will execute.
- When we use If word in real life ? If this happens then i do this
- Means If is followed by a condition and if that condition is true then we perform action
- Eg :
    - If (cash in hand is equal to zero)
      borrow money
    - If (age is 18)
      then valid voter.
- EX : When you login in insta or any social app then it ask for password if password is correct then you are a valid user , if pass is not correct then are not a valid user.
- It allows the computer to evaluate the expression first and then, depending on whether the value of the expression(relation or condition) is 'true' or 'false',
- It transfers the control to a particular statement.
- It has two paths to follow, 1 for true condition and 1 for false
- Syntax :
  If (Expression/ Condition)
  {
  ----
  block of code

  }
- Mostly we use relational or logical operators while checking condition with if block.

If- Else

- The if - Else statement is an extension of the simple if statement.
- Ex - we always have a backup plan If this is not happing then we go to this way.
- Ex - max of two there is only 1 ans if 1st no is not max then definatily 2nd is max
- Or switch or torch it can be on or off.

- If the condition is true (nonzero), the first statement is executed. If it is false (0), the second statement is executed.
- Eg :
    - If (marks is equal to or greater than passing marks)
      then pass
      else fail
- Syntax :
  If (Expression/ Condition)
  {

  If block of code

  }
  else
  {
  ------
  else block of code

  }
- EX : when you login in system do it gives you another chance for login so in else block we provide msg.
- If te test expression is true, then block statement,immediately following the if statement are executed;
- otherwise, the false block - statement are executed.
- only one block will be executed not the both.

Else if ladder

- if there are are multiple conditions on a single variable we can use else if ladder instead of nesting.
- ladder means steps one after other so if we want to check condition one after another that time
- Ex : If marks are above 85 then grade is A if marks are above 65 then grade is B etc.
- If a no is positive , negative, zero.
- Syntax :
  If (condition 1)
  {
  Statements 1
  }
  else if (condition 2)
  {
  statement 2
  }
  else if(condition 3)

{
statement 3
}
else
{
statement - x
}

- Ex: If the experience is less than 1 year, the designation is "Trainee."
- If the experience is between 1 and 3 years, the designation is "Junior Developer."
- If the experience is between 3 and 5 years, the designation is "Senior Developer."
- If the experience is more than 5 years, the designation is "Team Lead."
- if (theoryMarks >= 40) { // Check the first condition: theory marks if (practicalMarks >= 30)

Conditional Operator

- also known as ternary operator.
- Syntax : variable_name = exp_1 ? exp_2 : exp_3;
- if expression 1 is true then then exp2 is evaluated or exp 3 is evaluated.
- only 1 expression is evaluated.
- a = 5;
  b= 15;
  x= a>b ? a : b;

Nested If - Else

- Nested means able to be placed or stored one inside the other.
- The If inside another if then it is called as nested if.
- suppose we have to check multiple conditions.
- Nested if-else statements are like asking one question, and based on the answer, asking another question.
- Syntax:
  If (test condition 1)
  {
  If (test condition 2)
  {
  statement 1;
  }
  else
  {
  statement 2;

}
else
{
statement x;
}

- Ex : if user is male , if age is under 18 then he is boy else he is a man.
- EX : if username is correct then again check condition for password.
- Ex : Spotify - if a valid use? is a clg student ? yes then give discount not dont give like that.

Switch Case

- If we have to select one of the many alternatives, we can design program using if statements to control the selection.
- however such programs becomes to complex as number of alternatives increases. other way to handle such complex problem is by usin switch case statement.
- many times there is a need to execute some set of instructions depending on the option selected form the set of available choices.
- helps make decisions quickly when you have many options to choose from.
- Used when we have to control the selections.
- Think of it like a menu in a restaurant where you select one item.
- Ex: Coffe machine
- The switch statement is similar to else-if ladder statement as it provides multiple conditions.
- It tests the value of variable or expression against a series of different cases or values.
- If a match is found then the block of code is executed otherwise the default case is executed.
- A switch-case is like a series of interconnected doors in a corridor. Once you enter one door (case), unless explicitly stopped (with break), you might pass through the next ones automatically.
- Syntax :
  switch(expression)
  {
  case constant-1
  block-1;
  break;
  case constant-2
  block-2;
  break;

```
default:
default_block;
}
```

- Syntax consist of 2 parts - The declaration of switch i.e, the value upon which the test is to be made.
- list of cases stating which action to be taken if expression is match.
- default will execute if none of the case is matches.
- the result of expression is compared with case constants.
- you cannot give condition in statement
  case i>=20 gives error.
- float is not allowes in switch case.

Break Keyword

- break is a keyword which can break the lock and take the control outside that block of switch or loop.

Difference between Switch Case and If else Ladder

- If-else ladder statements evaluate conditions independently, while switch case statements have a built-in fall-through functionality.
- If-else ladder statements can compare values of all data types, while switch case statements can only compare integer and character values.
- If-else ladder statements can check for complex conditions, while switch case statements can only check for equality conditions.
- If-else ladder statements execute based on the condition inside the statement, while switch case statements execute based on the user's decision.

Iteration Control/ Loop control statements

- If we want to perform a action repeatedly multiple time that time we use iteration statements.
- Eg - In clg if you fail in a sub then whenever you dont get passing marks you are not eligilble for the later course.
- or snake ladder - until you are not on the finish point you continues playing.
- A certain set of instructions can be repeated for the specifies number of times or until a particular condition is statisfied.

- suppose we have to print hello world 100 times for print 8 table until num != 800.
- Every loop has four important parts.
    - **Body** - That contains set of instructions to be represented no of times.
    - **Initialization** - some variable like counter is initialized in initialization statement.
    - **Test Condition** - till this condition is true, loop wil be repeated. if condition is false, ececution is false, execution of loop will be stopped and execution continues with next statement after the loop.
    - **Modification** - during multiple execution of loop, some of the variables like counter must be modified.
- Iteration Statements are -
    - For
    - while
    - do while

while loop

- the situation where loop is repeated until any condition is false.
- Ex - LUDO : until you dont get 6 you cont move.
- Imagine an ATM asks you to enter your PIN until you provide the correct one. You get three attempts.
- Syntax :
  Initialization ;
  while (Condition)
  {
  body;
  modification;
  }
- typically, execution begins with initialization statement, and then checks the condition. if condition is true, body will be executed as well as variable will be modified.
- Once again condition will be checked and processis repeated. when condition will be false , control passes to the next statement after the loop.
- Suppose we want to print series from 1 to 10
- **Infinite loop** : The condition that is specified in the loop must get false after some finited number of executions, otherwise that loop goes into infinite state.
- **The Odd Loop** : In programming some times we dont know how many times we should repeat the loop. this situation can be programmed using odd loop concept.
    - while (ch=='y')
      what to enter another numbers Y/N.

Do-while

- In do-while loops, it first execute the body of the loop and then checks for the condition. if condition is true, body is repeated.

- difference is that the test in while is done before entering the loop and in do while it is made at the end of the loop and hence this loop will be executed at least before test is done at bottom.
- Mostly used with menu driven programs
- Syntax :

```
do
{
//body;
// modification;
} while(condition);
```

For Loop

- The for loop allow us to specify three things about loop in a single line.
- situation where loop is to be repeated fixed number of times.
- Syntax :

```
for(initialization; condition; modification)
{
// body ;
}
```

- OR
  Syntax :

```
initialization;
for(;condition;)
{
body;
modification;
}
```

Jump Statements/Instructions

- C also has some instructions that can be used to take the control directly to some other statement.
- such as break, continue, return and goto.
- ex : Flight, lift.
- Jump control statements are :
  - break;
  - continue;
  - goto;
  - return;

Break

- Many times there is a situations where control need to jump directily out of the loop, without waiting to get back to the test condition.
- Normally a loop is executed until the loop condition is true but sometime you may want to get out of the loop before the loop condition become false.
- this can be done using the break keyword inside the loop.
- ex: lift, while driving we take break.
- OR Imagine you're searching for a movie. Once you find the movie, you stop searching and leave the ott.

Continue

- There is a need of immediately continuing next iteration of the loop bypassing the remaining statements in the body of loop.
- Some times we may need to skio a part of the body of the loop and may need to go the next iteration of the loop without executing rest of the statement in the loop.
- skip the following statement and continue with the iteration.
- In the case of while and do-while loop, continue statement takes the cotrol to the tes condition.but in case of for loop, continue keyword take to the modification statements.
- You're sorting letters and skip opening those marked "spam." Instead, you move directly to the next letter.

goto

- the goto keyword is used to take control to some label statement in that function
- can be define any where in the function block.
- ex: You're filling out a form, and if you encounter an error, you're sent back to correct it at the specific error field.
-

return

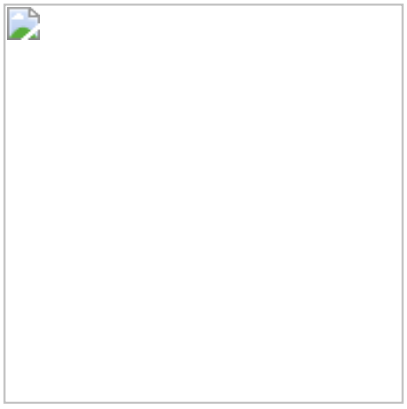
- return statement is used to end a function and send a value back to the part of the program that called the function.
- It helps a function "report back" with a result or simply exit without returning anything.
- Return the result of a calculation to the caller.

Dataypes

- Imagine you are storing water what you will use a bottle, a box , a tiffin, vessel.
- offcourse if you are storing storing water then you will use bottle
- and for daal or any other things you will use tiffin etc

- for all things to store data or things we required a variable or box to hold something.
- depend on the data we decide where to store it.
- and depending on our requirements we use datatypes means suppose i want to carry 1 ltr water for that can i use 5 ltr bottle no right ?
- means how space we reuired we use datatype baesd on it.
- Then What is dataype, Datatypes specifies the size and type of information the variable will store.

Types of datatype



Primary

- This stores basic kind of data like int, float, char.
- They are building blocks for more complex types.
- generally what type of data we use numeric, float or char although string also bt string is also created from char.

Derived

- derived data types are created from primary data types. but offers complex functionality.
- these are predefined in C bt allow programmer to handle more complex data and operation.

User-defined datatypes

- Are custom types created by programers using primary and derived data types.

Variables

- variables are names we given to storage area.
- suppose there is a int variable which stores int value, char variable store char data or value, float varible which stores float values or data.
- Syntax : datatype variable_name;

- ex : int a; int a, b; int a, b, c; float a char a,b;
- we tell compiler that allocate a memory for int datatypes
- like we say our frnd as take my seat or allocate it for me.
- we can also initiallize data by while declaring variables. - syntax : datatype varibale_name = value; - ex : int a = 10;

Rules for defining variables

- For defining names there are certain rules like for humans we cant give name as 123 or 54 or 123hello, there are certain rules same in C there are certain rule to follow like
    - A variable name must only contain alphabets, digits, and underscore.
    - A variable name must start with an alphabet or an underscore only. It cannot start with a digit.
    - No white space is allowed within the variable name.
    - A variable name must not be any reserved word or keyword.

FILE HANDLING IN C

What is a File?

- In normal data types, data is stored temporarily in memory (RAM).
- If we need to store data permanently, we use **files**.
- A file is a place on the disk where a group of related data is stored.
- Files help in data persistence and easy retrieval of information.

File Operations

- **Create**
- **Open** → `fopen()`
- **Read** → `fread()`, `fscanf()`
- **Write** → `fwrite()`, `fprintf()`
- **Close** → `fclose()`
- **Delete** → `remove()`
- **Rename** → `rename()`

File Structure

- `FILE` is an inbuilt struct used for handling files.
- The datatype for files in C **must be `FILE` (uppercase).**
- Example:

```
FILE *fptr;  // Pointer to a file
```

Opening a File (`fopen`)

- Used to open a file.
- Syntax:

```
fopen("file_name", "mode");
```

File Modes

Mode	Description
`r`	Read
`rb`	Read binary data
`w`	Write (creates new file if not exists)
`wb`	Write binary data
`a`	Append
`ab`	Append binary data
`r+`	Read + Write
`w+`	Write + Read (overwrites existing file)
`a+`	Append + Read

Closing a File (`fclose`)

- After reading/writing, always close the file using `fclose()`.
- Syntax:

```
fclose(fptr);
```

`fprintf` and `fscanf`

- Works like `printf` and `scanf`, but for files.
- **Syntax:**

```
fscanf(pointer, "format specifiers", &variable_list);
fprintf(pointer, "format specifiers", variable_list);
```

- **Example:**

```
FILE *fptr = fopen("data.txt", "w");
int num = 10;
fprintf(fptr, "%d", num);
fclose(fptr);
```

`fread` and `fwrite`

- Unlike `fscanf` and `fprintf`, these work in **binary mode**.
- Takes **4 parameters**:

```
fread(address_of_variable, no_of_bytes, no_of_records, file_pointer);
fwrite(address_of_variable, no_of_bytes, no_of_records, file_pointer);
```

Parameters:

1. **Address** → Address of variable to read/write.
2. **No. of bytes** → Size of data to be read/written (`sizeof()` operator is commonly used).
3. **No. of records** → Specifies how many records are processed in one function call.
4. **File pointer** → The file from which data is read/written.

Additional File Operations

Deleting a File (`remove`)

- Used to delete a file from the system.
- Syntax:

```
remove("file_name");
```

- Example:

```
if (remove("data.txt") == 0)
    printf("File deleted successfully");
else
    printf("Error deleting file");
```

Renaming a File (`rename`)

- Used to rename an existing file.
- Syntax:

```
rename("old_file_name", "new_file_name");
```

- Example:

```
if (rename("old.txt", "new.txt") == 0)
    printf("File renamed successfully");
else
    printf("Error renaming file");
```

This document provides a complete overview of file handling in C, including additional operations like deleting and renaming files.

Functions in C

What are Functions?

A function in C is a block of code that performs a specific task. Examples in real life include:

- A calculator that performs arithmetic operations.
- An air conditioner that cools a room.
- A TV that displays movies.

Types of Functions

1. **Predefined Functions**
    - These functions already have their definitions written in C libraries.
    - Examples: `printf()`, `scanf()`, `clrscr()`.
2. **User-Defined Functions**
    - These are functions where the programmer writes their own definition.
    - The user decides the parameters and return type.
    - Example:

```
int addition() {
    int a, b, res;
    res = a + b;
    return res;
}
```

User-Defined Functions

Why Use Functions?

- The execution of a program starts with the `main()` function.
- Writing all code in a single `main()` function makes programs large and complex.
- Debugging, testing, and maintaining such code becomes difficult.
- Functions break the code into smaller, manageable sections.

Syntax:

```
return_type function_name (parameters) {
    // Function body
}
```

- **return_type**: The datatype of the value the function returns.
- **function_name**: The name of the function.
- **parameters**: Input values passed to the function.
- Example: `int addition (int a, int b);`

Three Parts of a Function

1. **Declaration**
    - Functions must be declared before use to avoid "undefined function" errors.
    - Declaration is placed below `#include` statements.
    - Example:

      ```
      #include<stdio.h>
      int addNum(int a, int b);
      ```

2. **Definition**
    - Where the actual logic of the function is written.
3. **Call**
    - Function is executed when it is called in `main()`.

Function Types

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

    - Only control is passed to the function, nothing is returned.

2. **Function with Return Type but No Arguments**
    - A function that returns a value but does not take input.
3. **Function with No Return Type but Has Arguments**
    - A function that accepts input but does not return a value.
4. **Function with Return Type and Arguments**
    - A function that takes input and returns a value.

Recursive Functions

- A function that calls itself is called a **recursive function**.
- Example:

```
int factorial(int n) {
    if (n == 0) return 1;
    else return n * factorial(n - 1);
}
```

Passing Values to a Function

- The scope of a variable is limited to the function in which it is declared.
- Variables from one function cannot be used in another.
- The type, order, and number of actual and formal arguments must match.

Actual and Formal Arguments

```
void main() {
    function(arg1, arg2, arg3);  // Function Call (Actual Arguments)
}

int function(arg1, arg2, arg3) {  // Function Definition (Formal Arguments)
    // Function body
}
```

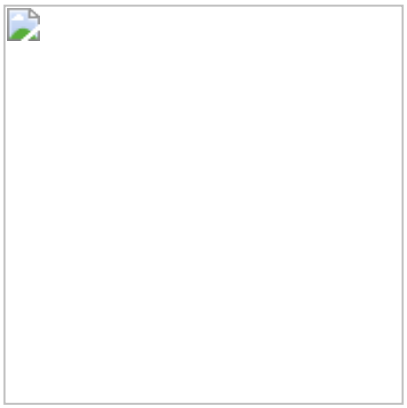
Printf

- printing message allows us to communicate with users, display result and present information clearly.
- printf stands for print formated

- syntax : printf("format string",variable); OR printf("Message");
    - format string : contains text and special placeholder for variables.
    - variables : These are the values we want to display, which will replace the placeholder in format string.
    - Message : convey msg to user.

Format Specifiers

- used to define the type of data you want to print
- format specifiers are placeholders, indicating what kind of value will be printed.
- Ex -you are introducing a friend how we do that, you will say
- this is [Name], and he is [Age] year old.
- here, [Name] & [Age] are placeholders for the actual name and age of your friend.
- these placeholders will be replaced with the actual info.
- Common Format Specifier



Escape Sequence

- When we write something on notebook or in notepad we do, we start a new paragraph then we move to next line, and give some space before paragraph.
- for doing such task c language provides escape sequences
- they are start with backslash and are followed by a specific character that tells the computer to perform a certain action

List of Escape Sequences

Escape Sequence	Name	Description
\a	Alarm or Beep	Generates a bell sound in the C program.
\b	Backspace	Moves the cursor one place backward.
\f	Form Feed	Moves the cursor to the start of the next logical page.
\n	New Line	Moves the cursor to the start of the next line.
\r	Carriage Return	Moves the cursor to the start of the current line.

Escape Sequence	Name	Description
\t	Horizontal Tab	Inserts whitespace to the left of the cursor accordingly.
\v	Vertical Tab	Inserts vertical space.
\\	Backslash	Inserts a backslash character.
\'	Single Quote	Displays a single quotation mark.
\"	Double Quote	Displays double quotation marks.
\?	Question Mark	Displays a question mark.
\ooo	Octal Number	Represents an octal number.
\xhh	Hexadecimal Number	Represents a hexadecimal number.
\0	NULL	Represents the NULL character.

Scanf()

- Used to take input from user.
- Scanf is define in Stdio.h header file
- Syntax : scanf("access_specifiers", address_of_variable);
- **access_specifiers** : Specifies the type of data you want to take input
- **Address_of_variable** : specifie the address of variable you want to store data.

What is programming Language

A programming language is a computer language that is used by programmers to communicate with computers. It is a set of instructions written in any specific language (C, C++, Java, Python) to perform a specific task.

Program

set of instructions which performs specific tasks.

Task : Listen Music

Instructions

1. open music application.
2. select your favourite song.
3. Enjoy music.

Task: Make a Cup of Tea

Instructions:

1. Light a gas.
2. Take a vessel.

3. Add water.
4. Add tea and sugar.
5. After boiling, add milk to it.
6. turn off gas

Computer follows each and every instructions.

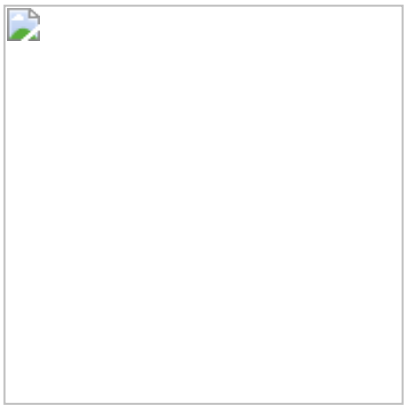
Language

Used by humans for communication.

Ex: Hindi, English, Marathi

A languages which is used by humans to communicate with computers are called programming languages.

Types of Languages



Low level Language

- low-level languages are programming languages that are closer to machine code and the hardware.
- Low-level languages include:
- Machine Language (or Machine Code) / Binary languages.

High level Language

- They are designed to be easier for humans to read, write, and maintain, allowing programmers to focus on solving problems rather than managing hardware details.
- In high level language there is pre-defined symbol by using that symbol we can create programs.
- high level language is converted in low level by using compiler or interpreter.
- High Level Languages include:

c++, java, python, .net

Middle level language

- Middle-level languages (MLLs) are programming languages that strike a balance between high-level languages (HLLs) and low-level languages (LLs), offering some advantages of both worlds.

- Middle-level languages allow developers to write code that is somewhat hardware-independent (like high-level languages) but also provides enough control over the hardware and memory to allow for optimizations (like low-level languages).

- Middle level languages include:

c, Assembly.

What is C Language

- A general-purpose, structured programming language.
- Combines low-level memory access with high-level language features.
- Designed for system programming like operating systems and compilers.
- C has greatly influenced many other programming languages, including C++, C#, Java, JavaScript, and Python.
- It introduced programming concepts like structured programming and pointers that became foundational in other languages.

Common Uses:

- **Operating Systems**: C was used to write UNIX, and it remains popular for OS development due to its low-level capabilities.

- **Embedded Systems**: Used in microcontrollers, IoT devices, and other hardware-level applications where control over memory and performance is essential.

- **Compilers and Interpreters**: Many compilers and interpreters for other languages are written in C, due to its efficiency and close-to-hardware control.

- **Game Development**: Due to its speed and efficiency, C is still used in game development, especially for systems where performance is critical.

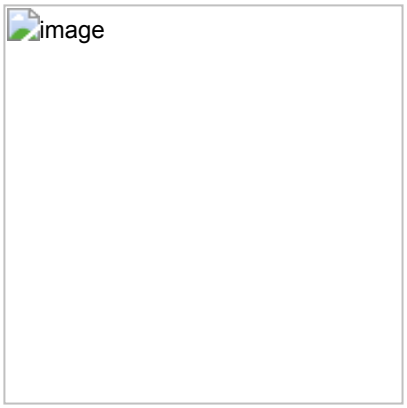
- **Database Management**: C has been used in the development of popular database management systems like MySQL and Oracle.

- **Advantages of Using C**:

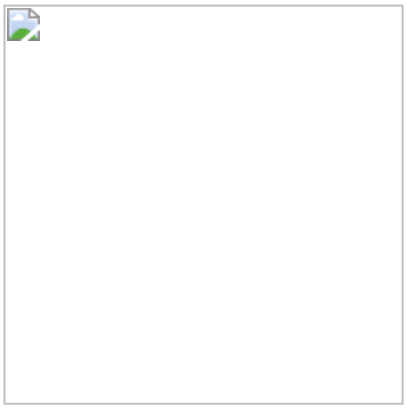
    - High performance and efficiency for system-level programming.
    - Flexibility to directly interact with hardware.
    - Strong community and extensive libraries.

History of the C Language

- **Origins**:
    - **Developed by**: Dennis Ritchie at Bell Labs
    - **Developed in**: 1972s
    - **Purpose**: Created primarily for system programming and to develop the UNIX operating system.
- **Evolution**:
    - **BCPL**: C evolved from an earlier language called BCPL (Basic Combined Programming Language), developed by Martin Richards in the 1960s.
    - **B Language**: Ken Thompson created the B language based on BCPL in 1969, which influenced C's design.
    - **Introduction of C**: Dennis Ritchie modified B to create C in 1972, adding features to improve performance and control over hardware, making it suitable for system-level programming.
- **UNIX and C**:
    - The UNIX operating system was initially written in assembly language but was rewritten in C in 1973.
    - Rewriting UNIX in C allowed for greater portability, making it easier to modify UNIX for different machines.
    - This established C as a powerful language for operating systems and laid the foundation for its widespread adoption.
- **Standardization**:
    - **ANSI C (C89)**: In 1989, the American National Standards Institute (ANSI) standardized C, creating ANSI C, also known as C89.
    - **ISO C (C90)**: The International Organization for Standardization (ISO) adopted the ANSI C standard in 1990, leading to further global use.
    - **C99**: Introduced in 1999 with new features such as inline functions, new data types (e.g., `long long int`), and flexible array members.
    - **C11**: In 2011, C11 introduced features to improve performance and compatibility, such as multithreading support, better Unicode support, and new libraries.
    - **C17 and C23**: Further updates have been introduced with minor improvements and optimizations to maintain C’s relevance.
- **Legacy and Influence**:
    - **Popularity**: C became widely popular in academia and industry, largely due to its efficiency, simplicity, and portability.
    - **Foundation for Other Languages**: C influenced many modern programming languages, including C++, Java, and Python, and remains foundational for system programming and embedded systems.
    - **Long-Lasting Relevance**: Despite being over 50 years old, C is still widely used for systems programming, embedded systems, and performance-critical applications.



How User Communicate with System



Features

- **Simple** : when we start learning bicycle once you know paddle, break and know how to balance then it is easy, same when we learn oprators, datatypes, variables, loops, etc then C is very simple language.
- **Portable** : when you write a C program on one machine and it can run on another machine with minimal or no changes.
- **Efficiency** : C programs are known for being efficient in terms of execution, speed and memory usage. C was used to build the unix operating system, which needs to be fast and resource Efficient.
- **Structured Language** : In C we break down a large problem into smaller and more manageable parts. This makes it easier to solve complex problems.
- **Rich Library** : These libraries contain code that you can use in your program without having to write it from scratch. It's like borrowing a ready made toolkit, instead of making every tool yourself.
- **Memory Management** : Think of memory Management like organizing your backpack for a trip. you decide what goes where so you can pack efficiently and access things quickly. In C language, we use functions like malloc() and free() to allocate ad free memory.

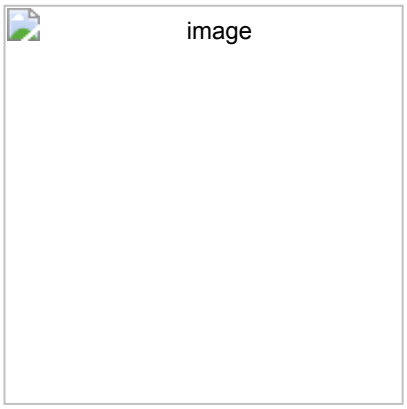
- **Fast Execution** : The code written in C is directily translated in machine code by a compiler. TRanslating a document directily from one language to another without any middleman, No time is wasted in the process. Thats why performance critical applications like game engines or database are written in C.

- **Pointers** : Pointers are variables that stores the memory address of another variable. Imagine if you had a map that pointed directily to a treasure, instead of having to search for it. Thats how pointer works direct access to memory.

- **Extensible** : C is an extensible, meaning you can add your own functions to the standard onces. If you're designing a car and you want to add special features like automatic lights or custom speakers.

keywords

- keywords are the words whose meaning has already assigned or explained to the computers.
- The keyword cannot be made as variable.
- the key words are also known as reserve word.
- Eg: In traffic signal there are fixed meaning of that colours, red means stop, green means go etc.
- OR Keywords are like grammar rules in a language. Just as grammar ensures meaningful communication, keywords ensure that the program is meaningful and executable.
- same like that keywords have special meanings for compilers.
- There are only 32 keywords available in 'c' but in turbo 'c' it is 58.

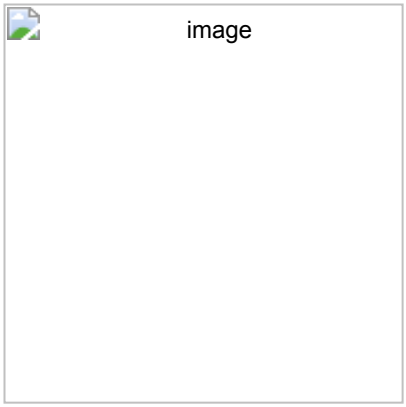


- Establish Standard Rules
- They provide a uniform syntax for writing code, making it easier for developers and compilers to understand and execute the program.
- Keywords eliminate the need for developers to create instructions from scratch for common tasks like looping, decision-making, or memory management.
- Instead of writing complex machine code, programmers can use keywords to convey the same functionality in a concise way.

Operators

- operators are symbols which perform some operations.
- An operator is a symbol that tells the computer to perform certain mathematical or logical manipulation.

- Operators are used in programs to manipulate data & variables i.e on operands
- they usually form a part of mathematical or logical exression.
- Operators are symbols or special words in C that tell the computer to do something with the data (like variables or numbers).
- They are like instructions to perform actions, such as adding numbers, comparing values, or making decisions.



Arithmatic

- Used to do basic math like addition, subtraction, multiplication, etc.

Relational

- Used to compare two things or relation between them and give a result in true or false.

Logical Operators (Decision-Making)

- Used to combine multiple conditions.

Assignment

- Used to assign values to variables.

Increment and Decrement Operators

- Used to increase or decrease a value by 1.

Conditional Operator

- The (? :) is a conditional operator.
- also know as ternary operator.
- exp1 ? exp2 : exp3;
- here the exp1 is evaluated first if it is non zero (true) then exp2 is evaluated othervise exp3 is evaluated and form the result of conditional exp.
- only one expresion either exp2 or exp3 is evaluated.

Pointers in C

Definition:

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

Real-Life Analogy:

Imagine a remote control (pointer) that controls a television (memory). Instead of going to the TV (direct variable access) every time to change channels, you use the remote (pointer) to perform the action more efficiently. By learning pointers, you're effectively gaining the "remote control" for managing data in your program.

Key Points:

- Also called an **address variable**.
- To store addresses of different types of variables, the pointer should be declared with the corresponding data type.
- The data type of a pointer can be `char`, `float`, `int`, etc.

Syntax:

```
char *ptr;   // Pointer to a character variable
```

Pointer Initialization:

```
int *ptr;    // Declare pointer
ptr = &a;    // Assign address of variable 'a' to pointer
```

Referencing and De-referencing

- **Referencing (`&`)**: Getting the address of a variable.
- **De-referencing (`*`)**: Accessing the value stored at the address held by the pointer.

Example:

```
int a = 10;
int *p = &a;

printf("Address of a: %p", &a);  // Reference: prints address of 'a'
printf("Address stored in pointer p: %p", p);  // Address in pointer
printf("Value at address stored in p: %d", *p);  // De-referencing: prints value of 'a'
```

Symbol	Meaning
`&p`	Address of variable `p`
`p`	Address stored in pointer `p`
`*p`	Value at the address stored in `p`

Pointers to Pointers

A pointer can store the address of another pointer, forming a **pointer to a pointer**.

Syntax:

```
int **ptr;
```

Example:

```
int a = 10;
int *p = &a;
int **pp = &p;  // Pointer to pointer

printf("Value of a: %d", a);
printf("Value using pointer p: %d", *p);
printf("Value using pointer to pointer pp: %d", **pp);
```

Pointers and Arrays

Pointers can be used to iterate through arrays efficiently.

Example:

```
int arr[5] = {1, 2, 3, 4, 5};
int *ptr = arr;

for(int i = 0; i < 5; i++) {
    printf("\n Address: %p", (ptr + i));  // Address of each element
    printf("\n Value: %d", *(ptr + i));   // Value at each address
}
```

C-Programming-

Strings

- The way a group of integers can be store in an integer array similarly a group of characters can be store in character array.
- character array are called as strings.
- string is that it is always terminated by special character '\0'.
- Eg: char name [] = {'H', 'e','l','l','o','\0'};
- OE char name [] = "Hello";
- '\0' is inserted by C it automatically.
- each char occupied one byte of memory.
- \0 look like 2 char bt they are 1 single char with the \ indicating what follows is something special.
- \0 is called as null character.
- it is imp character becoz it is the only way that compiler knows where strings ends.
- For traversing string we have to give condition as for(i=0; i!='\0'; i++)
- %S access specifier is used fro strings.
- char[]= "hello" this can store all characters specifies and char[5] can store only 5 characters.
- length of string

String Functions

- there are some inbuilt functions for performing operations on string
- all this built in functions are define in string.h file
- To use this functions, we have to include string.h header file
- String Functions are :
    1. strlen(str) -> returns numeric value
    2. strcat(tar,source) -> append 1 string at the end of another
    3. strcpy(tar,source) -> copys a string into other
    4. strcmp(str1, str2) -> campares 2 string
       returns 0 if strings are identical , if not then returns differnce between ascii values
    5. strlwr(str) -> returns string in lower case

6. strupr(str) -> returns str in upper case
7. strchr(str, n) -> finds first occurance of given character
8. strrev(str) -> reverse a string
9. gets(str) -> To take string input with whitespace
10. puts(str) -> print the string on console

Structure (`struct`)

Definition:

A `struct` in C is a user-defined data type that allows grouping variables of different data types together under one name. Each member of the structure has its own separate memory space.

Syntax:

```
struct StructName {
    dataType member1;
    dataType member2;
    ...
};
```

Key Points:

1. Each member has its own memory space.
2. The total size is the sum of all members' sizes.
3. All members can be accessed at the same time.
4. Useful when multiple variables need to store different values.

Union (`union`)

Definition:

A `union` in C is a user-defined data type similar to a structure but with a key difference— all members share the same memory space, meaning only one member can be stored at a time.

Syntax:

```
union UnionName {
    dataType member1;
    dataType member2;
    ...
};
```

Key Points:

1. All members share the same memory space.
2. The size of the union is equal to the largest member.
3. Only one member can be used at a time.
4. Useful when different types of data may be stored but not simultaneously.

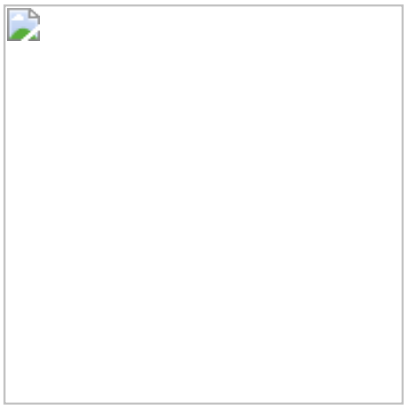
Difference Between Structure and Union

Feature	Structure	Union
Memory Allocation	Each member gets separate memory.	All members share the same memory.
Size	Sum of all members' sizes.	Size of the largest member.
Access	All members can be accessed simultaneously.	Only one member can be accessed at a time.
Usage	Used when multiple variables need to store different values.	Used when only one value is needed at a time to save memory.

Real-Life Example

1. **Classroom Seating (Structure)**:
    - In a classroom, each student gets their own chair, meaning memory is allocated separately for each.
    - This is similar to `struct`, where each member has its own space.
2. **Interview Chair (Union)**:
    - In an interview room, there is only one chair, and candidates sit one at a time.
    - This is like `union`, where different members share the same memory and only one can be used at a time.
3. **Application Form Example**:
    - Some applications ask for personal details like UID, PAN ID, or Driving License—only one is needed at a time.
    - But essential details like Name, Age, and DOB are compulsory.
    - Here, a `struct` can be used for mandatory fields, and a `union` for optional fields, optimizing memory usage.

Structure of C



1. Documentation

- Imagine you are assembling a piece of Robots or any game. They provide us the instructions manual that explains how to assemble it.
- Without the manual, it might be hard to understand how the piece fit together
- OR If you order food online they stick note on it. that's not so usefull but it's for the delivery boy to understand.
- This section includes comments, usually at the top of the program.
- It explains what the program does, who wrote it And other helpful information.
- Comments are ignored by compilers but are usefull for humans to understand the code.
- Ex - /* Program to print Hello */
- // date - 13-11-24.

2. Preprocessor Directives

- It's like checking a recipe before cooking, you gather all the ingredient needed for the dish
- Or like bringing a toolbar when building something.
- These are special lines that begin with # and tell the compilers to include external files or perform macros before the actual compilation starts.
- Common preprocessor directives is # include
- Which is used to include standard libraries like stdio.h
- Ex: #includ<stdio.h> OR #include<math.h>
- #include<stdio.h> lets you use printf() to display o/p and scanf() to get user i/p. without this, the program won't know how to handle i/p and o/p.
- Imagine you're about to bake a cake. Before you start baking, you first gather all the ingredients—sugar, flour, eggs. Similarly, before the program runs, we bring in ingredients, or in this case, libraries, by using #include.

3. Global Declaration Section

- Imagine you're in a group project, and you have a shared document that everyone can access.
- means its globally accessible.
- OR Best ex of this is a git.
- Imagine global variable as a "House key" shared by all family members. anyone in the house can use it.

4. Main() Function

- Main() Function as the central hub of a company All activities are co-ordinated from here.
- This is the entry point of program.
- Every program must have main() function this is where the execution of the program starts.
- inside main(), you declare variables write the logic and call other functions.
- Think of it like a school's morning assembly. Every student gathers here before heading to their individual classes. In the same way, the program gathers all its resources and starts running from the main() function.

5. User- Defined Functions

- These are the additional functions written by the programmers to perform specific tasks.
- If you go to the hospital and needs a specific check-up, then dr suggest you to visit specialist.
- user defined Functions help organize code, making it easier to read, debug and resue.

summary

Documentation

- This is like writting a note OR explaining the event purpose.
- like who is involved and what the goal is.

Preprocessor Directives

- These are like gathering the resources you need before starting the event.

Global Declaration

- you have a shared schedule that everyone can see & follow.

Main Function

- The main organizer Co-ordinates the event.
- they ask for the performance, schedule then & start the show.

User- Defined Functions

- These are the individual acts or performaces during the event.

- Each performer has a specific task, just like each func has a specific purpose in the program.