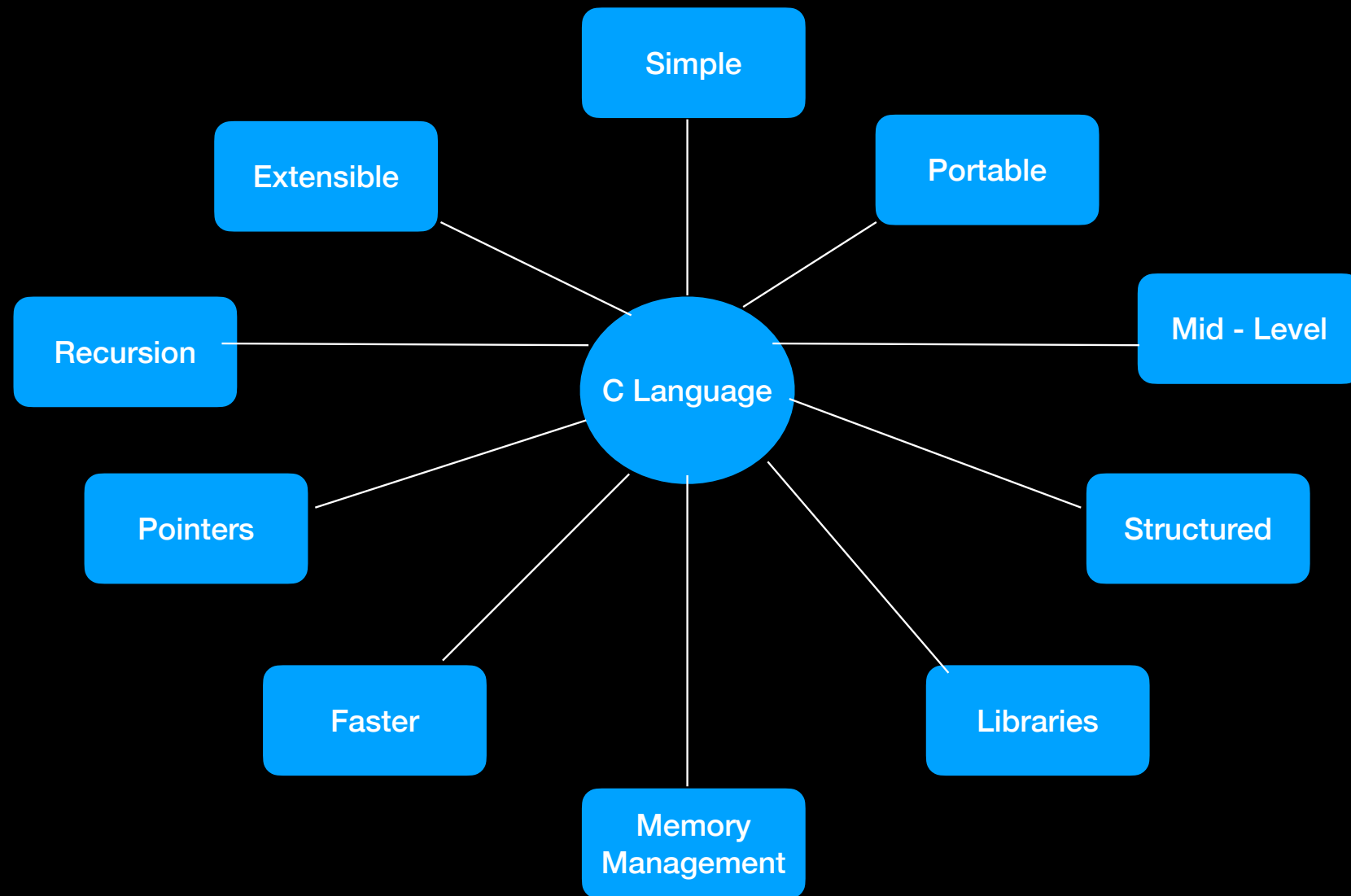


C-Language

History of C

- Who developed ?
- It was developed by Dennis Ritchie 1972 in Bell labs (At&T)
- What disadvantages?
 - ALGOL, B, BCPL are linear programming languages - Instructions are executed in linear format.
- Why it is developed?
 - disadvantages of B, BCPL
- What was it's actual purpose?
 - For Unix OS and C includes it's predecessors properties too.

Features of C



Data types in C

- Primitive Data types
- Void Types
- User defined Data types
- Derived Types

Data types in C

- Primitive types
 - These are arithmetic types
 - These include Integer, floating point, character
- Derived types
 - Data types that are derived from primitive Data types
- Void type
 - These types has no value and return nothing.
 - These come under primitive type.
- User defined Data types
 - These are used to assign names to constants, which make programs easy to read.

Data types in C

- Primitive Data types
 - Integer - int
 - Floating point - float
 - Double floating point - double
 - Character - char

Data types in C - Primitive DataTypes

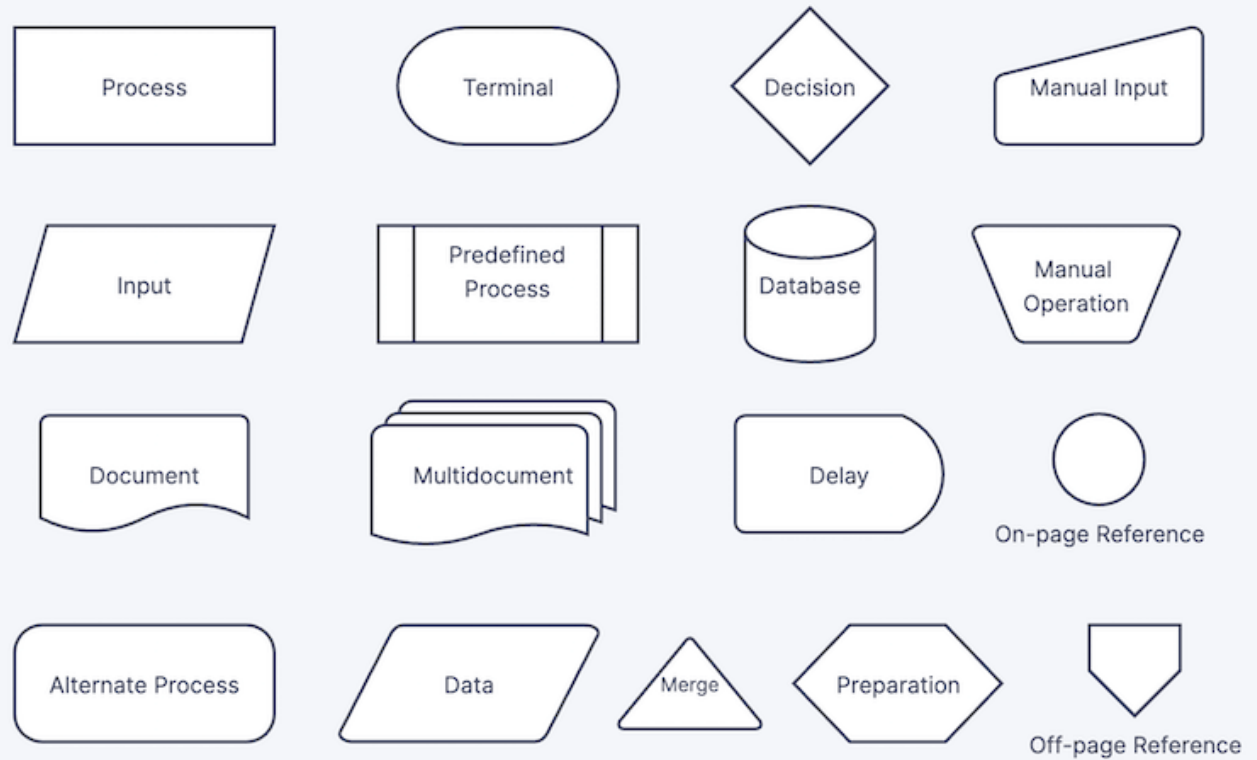
| Data Type | Memory (bytes) | Range | Format Specifier |
|------------------------|----------------|---------------------------------|------------------|
| short int | 2 | -32768 to 32767 | %hd |
| unsigned short int | 2 | 0 to 65,335 | %hu |
| unsigned int | 4 | 0 to 4,294,967,295 | %u |
| int | 4 | -2,147,483,648 to 2,147,483,647 | %d |
| long int | 4 | -2,147,483,648 to 2,147,483,647 | %ld |
| Unsigned long int | 4 | 0 to 4,294,967,295 | %lu |
| long long int | 8 | $-(2^{63})$ to $(2^{63})-1$ | %lld |
| unsigned long long int | 8 | 0 to 18,446,744,073,709,551,615 | %llu |
| Signed char | 1 | -128 to 127 | %c |
| unsigned char | 1 | 0 to 255 | %c |
| float | 4 | 1.2E-38 to 3.4E+38 | %f |
| double | 8 | 1.7E-308 to 1.7E+308 | %lf |
| long double | 16 | 3.4E-4932 to 1.1E+4932 | %Lf |

Data types in C - Primitive

- Write a program to print an Integer, float and a character
 - Print the size of a variable
- Let x be a variable assigned with 32767, what would be the output if we add one to it?
- Using character primitive type, print alphabets from a to z
- Using character primitive type print only consonants.

Flow Chart

A Flow chart is used for a pictorial representation of a logic.



Flow Chart and Algorithm

// Algorithm UserLogin

Function userLogin(username, password):

string user: username;

string pass: password;

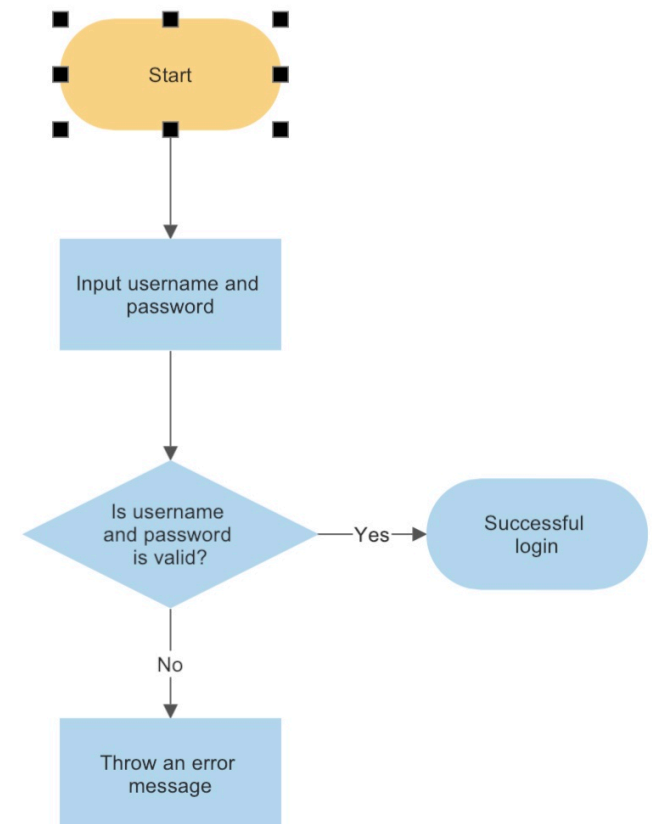
bool isValid : validateUser(user, pass)

if isValid = true

doLogin()

else

throw error;



Data types in C - Constants and Literals

- Constants are the fixed values that program may not alter during runtime.
- There are 2 different ways to define constants
 - Using define preprocessor
 - Const keyword

Data types in C - Constants challenge

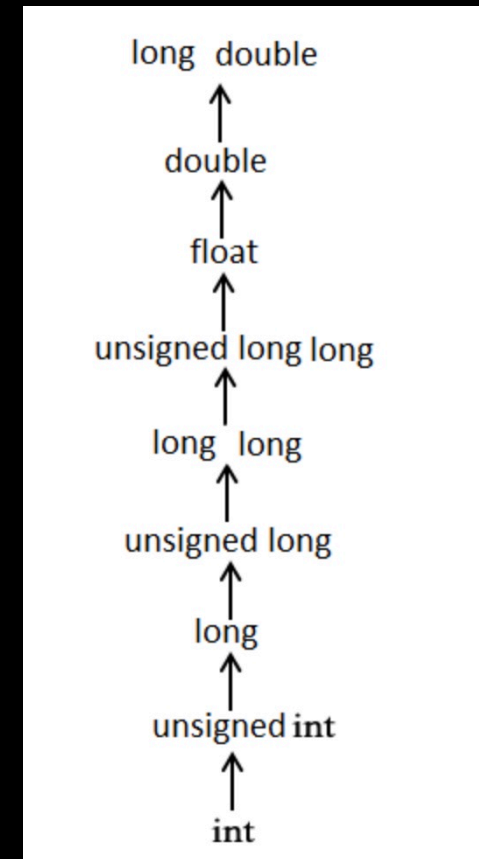
- Write a program to print area of a circle using
 - define preprocessor
 - Const keyword

Data types in C -Type casting

- Converting one data type to another data type is called Type casting.
- In typecasting, the destination data type may be smaller than the source data type when converting the data type to another data type, that's why it is also called narrowing conversion. For example, when we cast double (8) to int (4) we will be reducing the size of variable.
- There are 2 types of type casting
 - Implicit casting
 - Explicit Casting

Implicit Type casting

- Implicit casting is used to convert the data type of any variable without using the actual value that the variable holds.
- It performs the conversions without altering any of the values which are stored in the data variable. Conversion of lower data type to higher data type will occur automatically.
- Integer promotion will be performed first by the compiler. After that, it will determine whether two of the operands have different data types. Using the hierarchy in the diagram.



Explicit Type casting

- There are some cases where if the datatype remains unchanged, it can give incorrect output. In such cases, typecasting can help to get the correct output and reduce the time of compilation. In explicit type casting, we have to force the conversion between data types.
- In C programming, there are 5 built-in type casting functions.
 - **atof()**: This function is used for converting the string data type into a float data type.
 - **atol()**: This function is used for converting the string data type into a long data type.
 - **atoi()**: This data type is used to convert the string data type into an int data type
 - **ltoa()**: This function is used to convert the long data type into the string data type.
 - **itoba()**: This function is used to convert an int data type into a string data type.
- Functions like **ltoa** and **itoba** might not work in gcc and c99 compilers in linux OS, so there is a workaround to convert these numbers to strings
- We use **sprintf()** function to convert these types - `sprintf(stringVariable, FormatSpecifier, Number);` Number can be an int, long, float, double.

Keywords in C

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

Operators and Expressions- Arithmetic Operators

| Operator | Description | Example |
|----------|-----------------------|---------|
| + | Adds 2 operands | $A+B$ |
| - | Subtracts 2 operands | $A-B$ |
| * | Multiplies 2 operands | $A * B$ |
| / | Divides 2 operands | A/B |
| % | Modulus of 2 operands | $A\%B$ |
| ++ | Increment | $A++$ |
| -- | Decrement | $A--$ |

Operators and Expressions- Relational Operators

Relational operators are used to compare two values in C language. It checks the relationship between two values. If relation is true, it returns 1. However, if the relation is false, it returns 0.

| Operator | Description | Example |
|----------|--------------------------------|---------|
| = | Equal to(Assignment operator) | A = 10 |
| == | Is Equal to | A == B |
| != | Is Not Equal to | A != B |
| > | Is Greater than | A > B |
| < | Is Less than | A < B |
| >= | Is Greater than or equal | A >= B |
| <= | Is Less than or equal | A <= B |


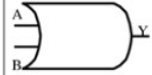

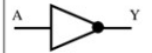

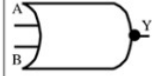

Operators and Expressions- Logical Operators

Logical operators are used to perform logical operations. It returns 0 or 1 based on the result of condition, whether its true or false. These operators are used for decision making.

| Operator | Description | |
|----------|-------------|---|
| && | Logical AND | Returns true if all operands are true |
| | Logical OR | Returns true if either one operand is true. |
| ! | Logical NOT | Used for logic inversion. |

Logic Gates

The table used to represent the boolean expression of a logic gate function is commonly called a **Truth Table**. A logic gate truth table shows each possible input combination to the gate or circuit with the resultant output depending upon the combination of these input(s).

| Basic Logic Gates | | | | | | | | | | | | | | | | | | | |
|-------------------|---|-----------------------------|--|--------------------|---|---|---|---|---|--|---|---|---|---|---|---|---|---|---|
| Logic | Schematic | Boolean Expression | Truth Table | English Expression | | | | | | | | | | | | | | | |
| AND |  | $A \cdot B = Y$ | <table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table> | A | B | Y | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | The only time the output is positive is when all the inputs are positive. |
| A | B | Y | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | | | | | | | | | | | | | | | | | |
| 0 | 1 | 0 | | | | | | | | | | | | | | | | | |
| 1 | 0 | 0 | | | | | | | | | | | | | | | | | |
| 1 | 1 | 1 | | | | | | | | | | | | | | | | | |
| OR |  | $A + B = Y$ | <table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table> | A | B | Y | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | The output will be positive when any one or all inputs are positive. |
| A | B | Y | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | | | | | | | | | | | | | | | | | |
| 0 | 1 | 1 | | | | | | | | | | | | | | | | | |
| 1 | 0 | 1 | | | | | | | | | | | | | | | | | |
| 1 | 1 | 1 | | | | | | | | | | | | | | | | | |
| XOR |  | $A \oplus B = Y$ | <table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table> | A | B | Y | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | The only time the output is positive is when the inputs are not the same. |
| A | B | Y | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | | | | | | | | | | | | | | | | | |
| 0 | 1 | 1 | | | | | | | | | | | | | | | | | |
| 1 | 0 | 1 | | | | | | | | | | | | | | | | | |
| 1 | 1 | 0 | | | | | | | | | | | | | | | | | |
| NOT |  | $\bar{A} = Y$ | <table><tr><th>A</th><th>Y</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table> | A | Y | 0 | 1 | 1 | 0 | The output is the opposite of the input. | | | | | | | | | |
| A | Y | | | | | | | | | | | | | | | | | | |
| 0 | 1 | | | | | | | | | | | | | | | | | | |
| 1 | 0 | | | | | | | | | | | | | | | | | | |
| NAND |  | $\overline{A \cdot B} = Y$ | <table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table> | A | B | Y | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | The output is positive provided all the inputs are not positive. |
| A | B | Y | | | | | | | | | | | | | | | | | |
| 0 | 0 | 1 | | | | | | | | | | | | | | | | | |
| 0 | 1 | 1 | | | | | | | | | | | | | | | | | |
| 1 | 0 | 1 | | | | | | | | | | | | | | | | | |
| 1 | 1 | 0 | | | | | | | | | | | | | | | | | |
| NOR |  | $\overline{A + B} = Y$ | <table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table> | A | B | Y | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | The only time the output is positive is when all the inputs are negative. |
| A | B | Y | | | | | | | | | | | | | | | | | |
| 0 | 0 | 1 | | | | | | | | | | | | | | | | | |
| 0 | 1 | 0 | | | | | | | | | | | | | | | | | |
| 1 | 0 | 0 | | | | | | | | | | | | | | | | | |
| 1 | 1 | 0 | | | | | | | | | | | | | | | | | |
| XNOR |  | $\overline{A \oplus B} = Y$ | <table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table> | A | B | Y | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | The only time the output is positive is when all the inputs are the same. |
| A | B | Y | | | | | | | | | | | | | | | | | |
| 0 | 0 | 1 | | | | | | | | | | | | | | | | | |
| 0 | 1 | 0 | | | | | | | | | | | | | | | | | |
| 1 | 0 | 0 | | | | | | | | | | | | | | | | | |
| 1 | 1 | 1 | | | | | | | | | | | | | | | | | |

Types of logic Gates

- AND
- OR
- XOR
- NOT
- NAND
- NOR
- XNOR