

C - Programming

Introduction to C

- The C Language is developed by **Dennis Ritchie** at the **Bell Telephone Laboratories in 1972** for creating system applications that directly interact with the hardware devices such as drivers, kernels, etc.
- C programming is considered as the base for other programming languages, that is why it is known as **mother language**.
- It can be defined by the following ways:
 1. Mother language
 2. System programming language
 3. Procedure-oriented programming language
 4. Structured programming language
 5. Mid-level programming language

C as a mother language

- C language is considered as the mother language of all the modern programming languages because **most of the compilers, JVMs, Kernels, etc. are written in C language**, and most of the programming languages follow C syntax, for example, C++, Java, C#, etc.
- It provides the core concepts like the array, strings, functions, file handling, etc. that are being used in many languages like C++, Java, C#, etc.

C as a system programming language

- A system programming language is used to create system software.
- C language is a system programming language because it **can be used to do low-level programming (for example driver and kernel)**.
- It is generally used to create hardware devices, OS, drivers, kernels, etc. For example, Linux kernel is written in C.
- It can't be used for internet programming like Java, .Net, PHP, etc.

C as a procedural language

- A procedure is known as a function, method, routine, subroutine, etc. A procedural language **specifies a series of steps for the program to solve the problem.**
- A procedural language breaks the program into functions, data structures, etc.
- C is a procedural language. In C, variables and function prototypes must be declared before being used.

C as a structured programming language



- A structured programming language is a subset of the procedural language.
- **Structure means to break a program into parts or blocks** so that it may be easy to understand.
- In the C language, we break the program into parts using functions. It makes the program easier to understand and modify.

C as a mid-level programming language

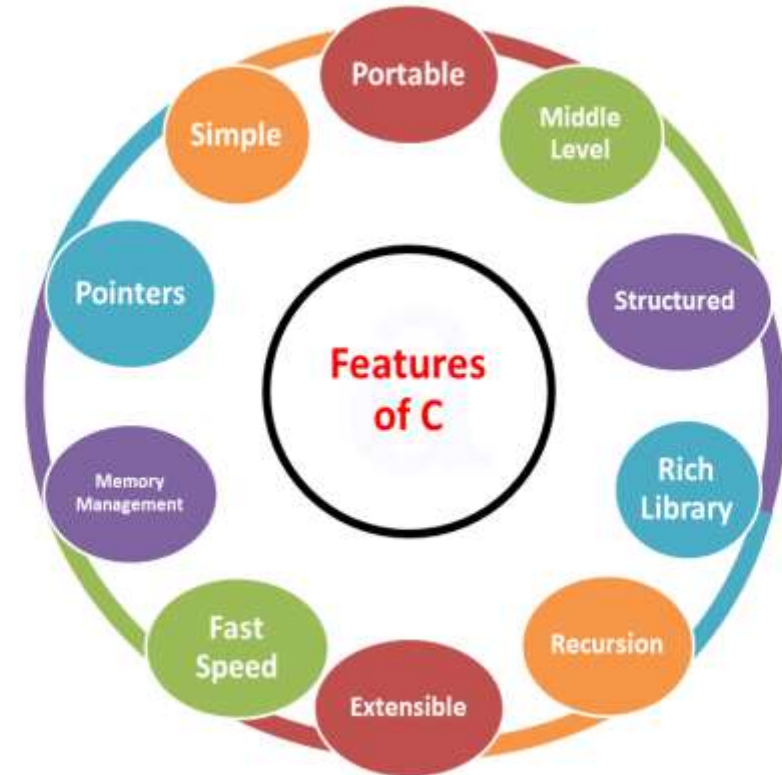


- C is considered as a middle-level language because it **supports the feature of both low-level and high-level languages**. C language program is converted into assembly code, it supports pointer arithmetic (low-level), but it is machine independent (a feature of high-level).
- A **Low-level language** is specific to one machine, i.e., machine dependent. It is machine dependent, fast to run. But it is not easy to understand.
- A **High-Level language** is not specific to one machine, i.e., machine independent. It is easy to understand.

Features of C Language

C is the widely used language. It provides many **features** they are

- 1.Simple
- 2.Machine Independent or Portable
- 3.Mid-level programming language
- 4.Structured programming language
- 5.Rich Library
- 6.Memory Management
- 7.Fast Speed
- 8.Pointers
- 9.Recursion
- 10.Extensible



1) Simple

C is a simple language in the sense that it provides a **structured approach** (to break the problem into parts), **the rich set of library functions, data types, etc**

2) Machine Independent or Portable

Unlike assembly language, c programs **can be executed on different machines** with some machine specific changes. Therefore, C is a machine independent language.

3) Mid-level programming language

C is **intended to do low-level programming**. It is used to develop system applications such as kernel, driver, etc. It **also supports the features of a high-level language**. That is why it is known as mid-level language.

4) Structured programming language

C is a structured programming language in the sense that **we can break the program into parts using functions**. So, it is easy to understand and modify. Functions also provide code reusability.

5) Rich Library

C **provides a lot of inbuilt functions** that make the development fast.

6) Memory Management

It supports the feature of **dynamic memory allocation**. In C language, we can free the allocated memory at any time by calling the **free()** function.

7) Speed

The compilation and execution time of C language is fast since there are lesser inbuilt functions and hence the lesser overhead.

8) Pointer

C provides the feature of pointers. We can directly interact with the memory by using the pointers. We **can use pointers for memory, structures, functions, array, etc.**

9) Recursion

In C, we **can call the function within the function.** It provides code reusability for every function. Recursion enables us to use the approach of backtracking

10) Extensible

C language is extensible because it **can easily adopt new features.**

First C Program

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

```
int main(){
```

```
printf("Hello C Language");
```

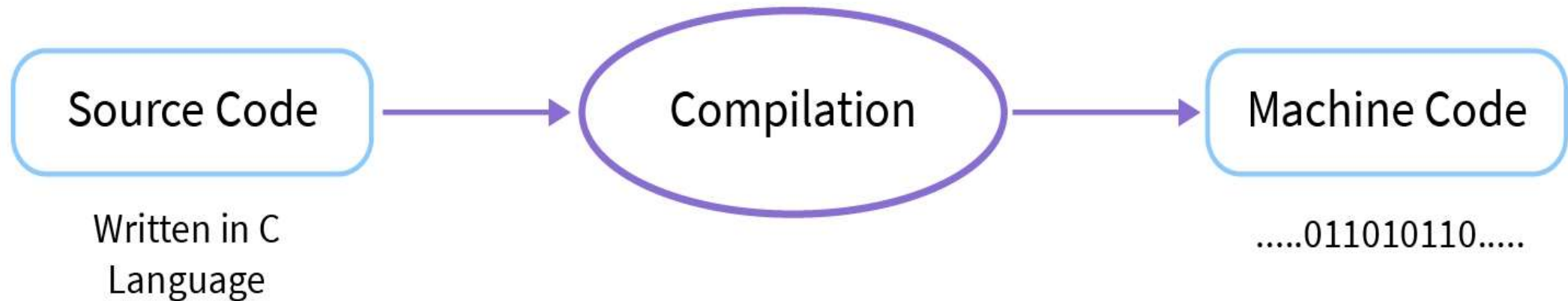
```
return 0;
```

```
}
```

- **#include <stdio.h>** includes the **standard input output** library functions.
- The **printf()** function is defined in **stdio.h**
- **int main()** The **main() function is the entry point of every program** in c language.
- **printf()** The **printf()** function is **used to print data** on the console.
- **return 0** The **return 0** statement, returns execution status to the OS. The 0 value is used for successful execution and 1 for unsuccessful execution.

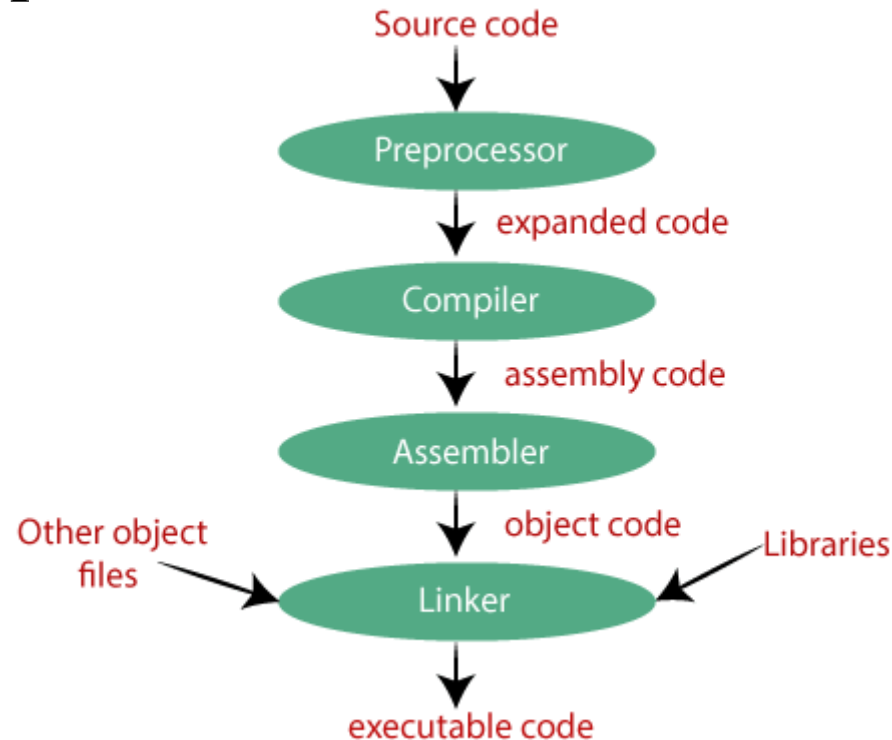
Compilation process in c

- The compilation is a process of converting the source code into object code. It is done with the help of the compiler.
- The compiler checks the source code for the syntactical or structural errors, and if the source code is error-free, then it generates the object code.



- The c compilation process converts the source code taken as input into the object code or machine code.
- The compilation process can be divided into four steps, they are

- **Preprocessor**
- **Compiler**
- **Assembler**
- **Linker**



Preprocessor

The source code is the code which is written in a text editor and the source code file is given an extension ".c". This source code is first passed to the preprocessor, and then the preprocessor expands this code. After expanding the code, the expanded code is passed to the compiler.

Compiler

The code which is expanded by the preprocessor is passed to the compiler. The compiler converts this code into assembly code. Or we can say that the C compiler converts the pre-processed code into assembly code.

Assembler

The assembly code is converted into object code by using an assembler. The name of the object file generated by the assembler is the same as the source file. The extension of the object file in DOS is '.obj,' and in UNIX, the extension is 'o'. If the name of the source file is '**hello.c**', then the name of the object file would be 'hello.obj'.

Linker

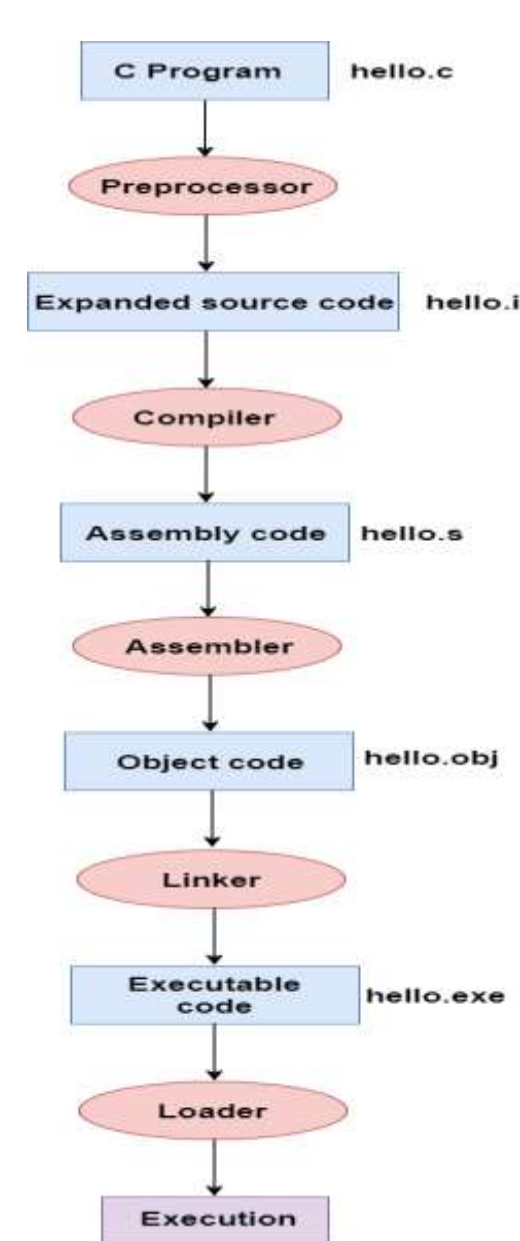
Mainly, all the programs written in C use library functions. These library functions are pre-compiled, and the object code of these library files is stored with '.lib' (or '.a') extension.

The main working of the linker is to combine the object code of library files with the object code of our program. Sometimes the situation arises when our program refers to the functions defined in other files; then linker plays a very important role in this. It links the object code of these files to our program. Therefore, we conclude that the job of the linker is to link the object code of our program with the object code of the library files and other files. The output of the linker is the executable file. The name of the executable file is the same as the source file but differs only in their extensions. In DOS, the extension of the executable file is '.exe', and in UNIX, the executable file can be named as 'a.out'. For example, if we are using printf() function in a program, then the linker adds its associated code in an output file.

Let's understand through an example.

hello.c

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



The following steps are taken to execute a program:

- Firstly, the input file, i.e., **hello.c**, is passed to the preprocessor, and the preprocessor converts the source code into expanded source code. The extension of the expanded source code would be **hello.i**.
- The expanded source code is passed to the compiler, and the compiler converts this expanded source code into assembly code. The extension of the assembly code would be **hello.s**.
- This assembly code is then sent to the assembler, which converts the assembly code into object code.
- After the creation of an object code, the linker creates the executable file. The loader will then load the executable file for the execution.

printf() and scanf() in C

- The printf() and scanf() functions are used for input and output in C language. Both functions are inbuilt library functions, defined in stdio.h (header file).

printf() function :

- The **printf() function** is used for output. It prints the given statement to the console
- The syntax of printf() function is :

printf("format string", argument_list);

The **format string** can be %d (integer), %c (character), %s (string), %f (float) etc.

scanf() function

- The **scanf()** function is used for input. It reads the input data from the console.
- The syntax of **scanf()** function is :
scanf("format string",argument_list);

Variables in C

- A **variable** is a name of the memory location.
- It is used to store data. Its value can be changed, and it can be reused many times.
- It is a way to represent memory location through symbol so that it can be easily identified.
- The syntax to declare a variable is: **type variable_list;**

Example:

1. `int a;`
2. `float b;`
3. `char c;`

Here, a, b, c are variables. The int, float, char are the data types.

We can also provide values while declaring the variables as given below:

1. `int a=10,b=20; //declaring 2 variable of integer type`
2. `float f=20.8;`
3. `char c='A';`

Rules for defining variables

- A variable can have alphabets, digits, and underscore.
- A variable name can start with the alphabet, and underscore only. It can't start with a digit.
- No whitespace is allowed within the variable name.
- A variable name must not be any reserved word or keyword, e.g. int, float, etc

Examples for valid variable names:

1. `int a;`
2. `int _ab;`
3. `int a30;`

Types of Variables in C

There are many types of variables in c:

1. local variable
2. global variable
3. static variable
4. automatic variable
5. external variable

Local Variable

- A variable that is declared inside the function or block is called a local variable.
- It must be declared at the start of the block.
- The variables must be initialize before it is used.

```
void function1(){  
    int x=10; //local variable  
}
```


Global Variable

- A variable that is declared outside the function or block is called a global variable.
- Any function can change the value of the global variable.
- It is available to all the functions.
- It must be declared at the start of the block

```
int value=20; //global variable  
void function1(){  
    int x=10; //local variable  
}
```

Static Variable

- A variable that is declared with the static keyword is called static variable.
- It retains its value between multiple function calls.

```
void function1(){  
    int x=10;//local variable  
    static int y=10;//static variable  
    x=x+1;  
    y=y+1;  
    printf("%d,%d",x,y);  
}
```

- If you call this function many times, the **local variable will print the same value** for each function call, e.g, 11,11,11 and so on. But the **static variable will print the incremented value** in each function call, e.g. 11, 12, 13 and so on.

Automatic Variable

- All variables in C that are declared inside the block, are automatic variables by default. We can explicitly declare an automatic variable using **auto keyword**.

```
void main(){  
    int x=10;//local variable (also automatic)  
    auto int y=20;//automatic variable  
}
```

External Variable

- We can share a variable in multiple C source files by using an external variable. To declare an external variable, you need to use **extern** keyword.

myfile.h

```
extern int x=10;//external variable (also global)
```

program1.c

```
#include "myfile.h"
```

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

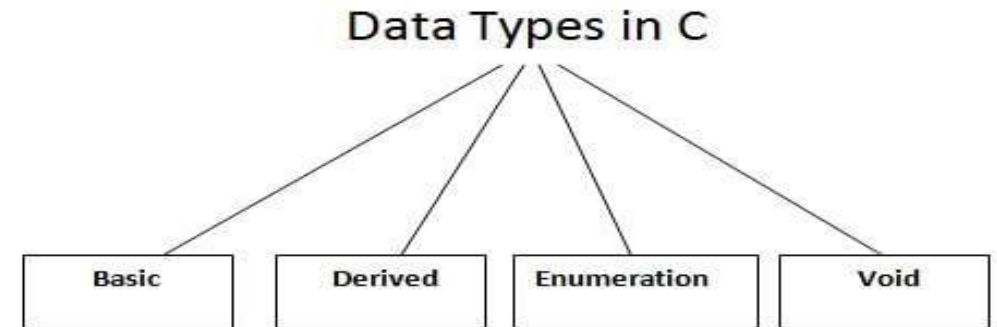
```
void printValue(){
```

```
    printf("Global variable: %d", global_variable);
```

```
}
```

Data Types in C

- A data type specifies the type of data that a variable can store such as integer, floating, character, etc.
- There are the following data types in C language



Types	Data Types
Basic Data Type	int, char, float, double
Derived Data Type	array, pointer, structure, union
Enumeration Data Type	enum
Void Data Type	void

Basic Data Types

- The basic data types are integer-based and floating-point based.
- C language supports both signed and unsigned literals.
- The memory size of the basic data types may change according to 32 or 64-bit operating system.

Int:

- **Integers** are entire numbers without any fractional or decimal parts, and the **int data type** is used to represent them.
- It is frequently applied to variables that include **values**, such as **counts**, **indices**, or other numerical numbers.
- The **int data type** may represent both **positive** and **negative numbers** because it is signed by default.
- An **int** takes up **4 bytes** of memory on most devices, allowing it to store values between around -2 billion and +2 billion.

Char:

- Individual characters are represented by the *char data type*. Typically used to hold *ASCII* or *UTF-8 encoding scheme characters*, such as *letters*, *numbers*, *symbols*, or *commas*.
- There are *256 characters* that can be represented by a single char, which takes up one byte of memory.
- Characters such as '*A*', '*b*', '*5*', or '*\$*' are enclosed in single quotes.

Float:

- To represent integers, use the *floating data type*. Floating numbers can be used to represent fractional units or numbers with decimal places.
- The *float type* is usually used for variables that require very good precision but may not be very precise.
- It can store values with an accuracy of about *6 decimal places* and a range of about *3.4×10^{38}* in *4 bytes* of memory.

Double:

- Use two data types to represent **two floating integers**. When additional precision is needed, such as in scientific calculations or financial applications, it provides greater accuracy compared to float.
- **Double type**, which uses **8 bytes** of memory and has an accuracy of about **15 decimal places**, **yields larger values**. C treats floating point numbers as doubles by default if no explicit type is supplied.

```
int age = 25;
```

```
char grade = 'A';
```

```
float temperature = 98.6;
```

```
double pi = 3.14159265359;
```

- In the example above, we declare four variables: an **int variable** for the person's age, a **char variable** for the student's grade, a **float variable** for the temperature reading, and **double variables** for the **number pi**.

Derived Data Type

- Beyond the fundamental data types, C also supports **derived data types**, including **arrays**, **pointers**, **structures**, and **unions**. These data types give programmers the ability to handle heterogeneous data, directly modify memory, and build complicated data structures.

Array:

- An **array**, a **derived data type**, lets you store a sequence of **fixed-size elements** of the same type. It provides a mechanism for joining multiple targets of the same data under the same name.
- The index is used to access the elements of the array, with a **0 index** for the first entry. The size of the array is fixed at declaration time and cannot be changed during program execution. The array components are placed in adjacent memory regions.

```
#include <stdio.h>

int main() {
    int numbers[5]; // Declares an integer array with a size of 5 elements
    // Assign values to the array elements
    numbers[0] = 10;
    numbers[1] = 20;
    numbers[2] = 30;
    numbers[3] = 40;
    numbers[4] = 50;
    // Display the values stored in the array
    printf("Values in the array: ");
    for (int i = 0; i < 5; i++) {
        printf("%d ", numbers[i]);
    }
    printf("\n");
    return 0;
}
```

OUTPUT: Values in the array: 10 20 30 40 50

Pointer:

- A pointer is a derived data type that keeps track of another data type's memory address. When a pointer is declared, the data type it refers to is stated first, and then the variable name is preceded by an asterisk (*).
- You can have incorrect access and change the value of variable using pointers by specifying the memory address of the variable. Pointers are commonly used in tasks such as function pointers, data structures, and dynamic memory allocation.

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

```
int main() {  
    int num = 42;    // An integer variable  
    int *ptr;        // Declares a pointer to an integer  
  
    ptr = #          // Assigns the address of 'num' to the pointer  
  
    // Accessing the value of 'num' using the pointer  
    printf("Value of num: %d\n", *ptr);  
  
    return 0;  
}
```

Output: Value of num: 42

Structure:

- A structure is a derived data type that enables the creation of composite data types by allowing the grouping of many data types under a single name. It gives you the ability to create your own unique data structures by fusing together variables of various sorts.
 1. A structure's members or fields are used to refer to each variable within it.
 2. Any data type, including different structures, can be a member of a structure.
 3. A structure's members can be accessed by using the dot (.) operator.



```
#include <stdio.h>
#include <string.h>
// Define a structure representing a person
struct Person {
    char name[50];
    int age;
    float height; };
int main() {
    // Declare a variable of type struct Person
    struct Person person1;
    // Assign values to the structure members
    strcpy(person1.name, "John Doe");
    person1.age = 30;
    person1.height = 1.8;
    // Accessing the structure members
    printf("Name: %s\n", person1.name);
    printf("Age: %d\n", person1.age);
    printf("Height: %.2f\n", person1.height);
    return 0;
}
```

Output:

Name: John Doe

Age: 30

Height: 1.80

Union:

- A derived data type called a union enables you to store various data types in the same memory address.
- In contrast to structures, where each member has a separate memory space, members of a union all share a single memory space. A value can only be held by one member of a union at any given moment. When you need to represent many data types interchangeably, unions come in handy.
- Like structures, you can access the members of a union by using the dot (.) operator.

```
#include <stdio.h>

// Define a union representing a numeric value
union NumericValue {
    int intValue;
    float floatValue;
    char stringValue[20];
};

int main() {
    // Declare a variable of type union NumericValue
    union NumericValue value;

    // Assign a value to the union
    value.intValue = 42;

    // Accessing the union members
    printf("Integer Value: %d\n", value.intValue);

    // Assigning a different value to the union
    value.floatValue = 3.14;

    // Accessing the union members
    printf("Float Value: %.2f\n", value.floatValue);

    return 0;
}
```

Output:

Integer Value: 42

Float Value: 3.14

Enumeration Data Type

A set of named constants or **enumerators** that represent a collection of connected values can be defined in C using the **enumeration data type (enum)**. **Enumerations** give you the means to give names that make sense to a group of integral values, which makes your code easier to read and maintain.

```
#include <stdio.h>
// Define an enumeration for days of the week
enum DaysOfWeek {
    Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday };
int main() {
    // Declare a variable of type enum DaysOfWeek
    enum DaysOfWeek today;

    // Assign a value from the enumeration
    today = Wednesday;

    // Accessing the enumeration value
    printf("Today is %d\n", today);

    return 0;
}
```

Output: Today is 2

Void Data Type

The **void data type** is helpful for defining functions that don't accept any arguments when working with generic pointers or when you wish to signal that a function doesn't return a value. It is significant to note that while **void*** can be used to build generic pointers, void itself cannot be declared as a variable type.

Keywords in C

- A keyword is a **reserved word**. You cannot use it as a variable name, constant name, etc.
- There are only 32 reserved words (keywords) in the C language.

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

C Identifiers

- Identifier refers to name given to entities such as variables, functions, structures etc.
- Identifiers must be unique. They are created to give a unique name to an entity to identify it during the execution of the program.

For example:

```
int money;
```

```
double accountBalance;
```

Here, **money** and **accountBalance** are identifiers.

- Also remember, identifier names must be different from keywords. You cannot use `int` as an identifier because `int` is a keyword.

Rules for naming identifiers

- The first character of an identifier should be either an alphabet or an underscore, and then it can be followed by any of the character, digit, or underscore.
- It should not begin with any numerical digit.
- In identifiers, both uppercase and lowercase letters are distinct. Therefore, we can say that identifiers are case sensitive.
- Commas or blank spaces cannot be specified within an identifier.
- Keywords cannot be represented as an identifier.
- The length of the identifiers should not be more than 31 characters.
- Identifiers should be written in such a way that it is meaningful, short, and easy to read.

Types of identifiers

- Internal identifier
- External identifier

1. Internal Identifier

- If the identifier is not used in the external linkage, then it is known as an internal identifier.
- The internal identifiers can be local variables.

2. External Identifier

- If the identifier is used in the external linkage, then it is known as an external identifier.
- The external identifiers can be function names, global variables.

C Comments

In programming, comments are hints that a programmer can add to make their code easier to read and understand.

Types of Comments

There are two ways to add comments in C:

1. `//` - Single Line Comment
2. `/*...*/` - Multi-line Comment

1. Single-line Comments in C

- In C, a single line comment starts with //. It starts and ends in the same line.

For example:

```
#include <stdio.h>
int main() {
    // create integer variable
    int age = 25;
    // print the age variable
    printf("Age: %d", age);
    return 0;
}
```

2. Multi-line Comments in C

- In C programming, there is another type of comment that allows us to comment on multiple lines at once, they are multi-line comments.
- To write multi-line comments, we use the `/*....*/` symbol.

For example:

```
/* This program takes age input from the user  
It stores it in the age variable  
And, print the value using printf() */
```

```
#include <stdio.h>  
int main() {  
    int age;  
    printf("Enter the age: ");  
    scanf("%d", &age);  
    printf("Age = %d", age);  
    return 0;  
}
```

Use of Comments in C

1. Make Code Easier to Understand

- If we write comments on our code, it will be easier to understand the code in the future. Otherwise you will end up spending a lot of time looking at our own code and trying to understand it.
- Comments are even more important if you are working in a group. It makes it easier for other developers to understand and use your code.

2. Using Comments for debugging

- While debugging there might be situations where we don't want some part of the code. So, instead of removing the code we can simply convert them into comments.

C - Operators

- An operator is a symbol that tells the compiler to perform specific mathematical or logical functions.
- C language is rich in built-in operators and provides the following types of operators

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Other Operators

Operators in C

	Operators	Type
Unary Operator →	++, --	Unary Operator
Binary Operator {	+, -, *, /, %	Arithmetic Operator
	<, <=, >, >=, ==, !=	Relational Operator
	&&, , !	Logical Operator
	&, , <<, >>, ~, ^	Bitwise Operator
	=, +=, -=, *=, /=, %=	Assignment Operator
Ternary Operator →	?:	Ternary or Conditional Operator

Arithmetic Operators

- Assume variable **A** holds 10 and variable **B** holds 20

Operator	Description	Example
+	Adds two operands.	$A + B = 30$
-	Subtracts second operand from the first.	$A - B = -10$
*	Multiplies both operands.	$A * B = 200$
/	Divides numerator by denominator.	$B / A = 2$
%	Modulus Operator and remainder of after an integer division.	$B \% A = 0$
++	Increment operator increases the integer value by one.	$A++ = 11$
--	Decrement operator decreases the integer value by one.	$A-- = 9$

C Assignment Operators

- An assignment operator is used for assigning a value to a variable. The most common assignment operator is

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

C Relational Operators

- A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.
- Relational operators are used in decision making and loops.

Operator	Meaning of Operator	Example
==	Equal to	<code>5 == 3</code> is evaluated to 0
>	Greater than	<code>5 > 3</code> is evaluated to 1
<	Less than	<code>5 < 3</code> is evaluated to 0
!=	Not equal to	<code>5 != 3</code> is evaluated to 1
>=	Greater than or equal to	<code>5 >= 3</code> is evaluated to 1
<=	Less than or equal to	<code>5 <= 3</code> is evaluated to 0

C Logical Operators

- An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false.
- Logical operators are commonly used in decision making in C programming.

Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression <code>((c==5) && (d>5))</code> equals to 0.
	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression <code>((c==5) (d>5))</code> equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression <code>!(c==5)</code> equals to 0.

C Bitwise Operators

- Bitwise operators are used in C programming to perform bit-level operations.

Operators	Meaning of operators
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Other Operators

Comma Operator

- Comma operators are used to link related expressions together.

For example:

```
int a, c = 5, d;
```

The sizeof operator

- The sizeof is a unary operator that returns the size of data (constants, variables, array, structure, etc).

ternary operator ?:

reference operator &

dereference operator *

member selection operator ->

C Ternary Operator

- We use the ternary operator in C to run one code when the condition is true and another code when the condition is false.

Syntax of Ternary Operator

- The syntax of ternary operator is :

testCondition ? expression1 : expression 2;

- The **testCondition** is a boolean expression that results in either true or false.
If the condition is
 true - expression1 (before the colon) is executed
 false - expression2 (after the colon) is executed
- The ternary operator takes 3 operands (condition, expression1 and expression2). Hence, the name ternary operator.

Example: C Ternary Operator

```
#include <stdio.h>

int main() {
    int age;
    // take input from users
    printf("Enter your age: ");
    scanf("%d", &age);
    // ternary operator to find if a person can vote or not
    (age >= 18) ? printf("You can vote") : printf("You cannot vote");
    return 0;
}
```

C Flow Control Statements

- Control statements serve to control the execution flow of a program.
- Control statements in C are classified into selection statements, iteration statements, and jump statements.
- Selection statements serve to execute code based on a certain circumstance.
- Iteration statements loop through a code block until a condition is met.
- Jump statements serve to move control from one section of a program to another.
- The if statement, for loop, while loop, switch statement, break statement, and continue statement are C's most widely used control statements.

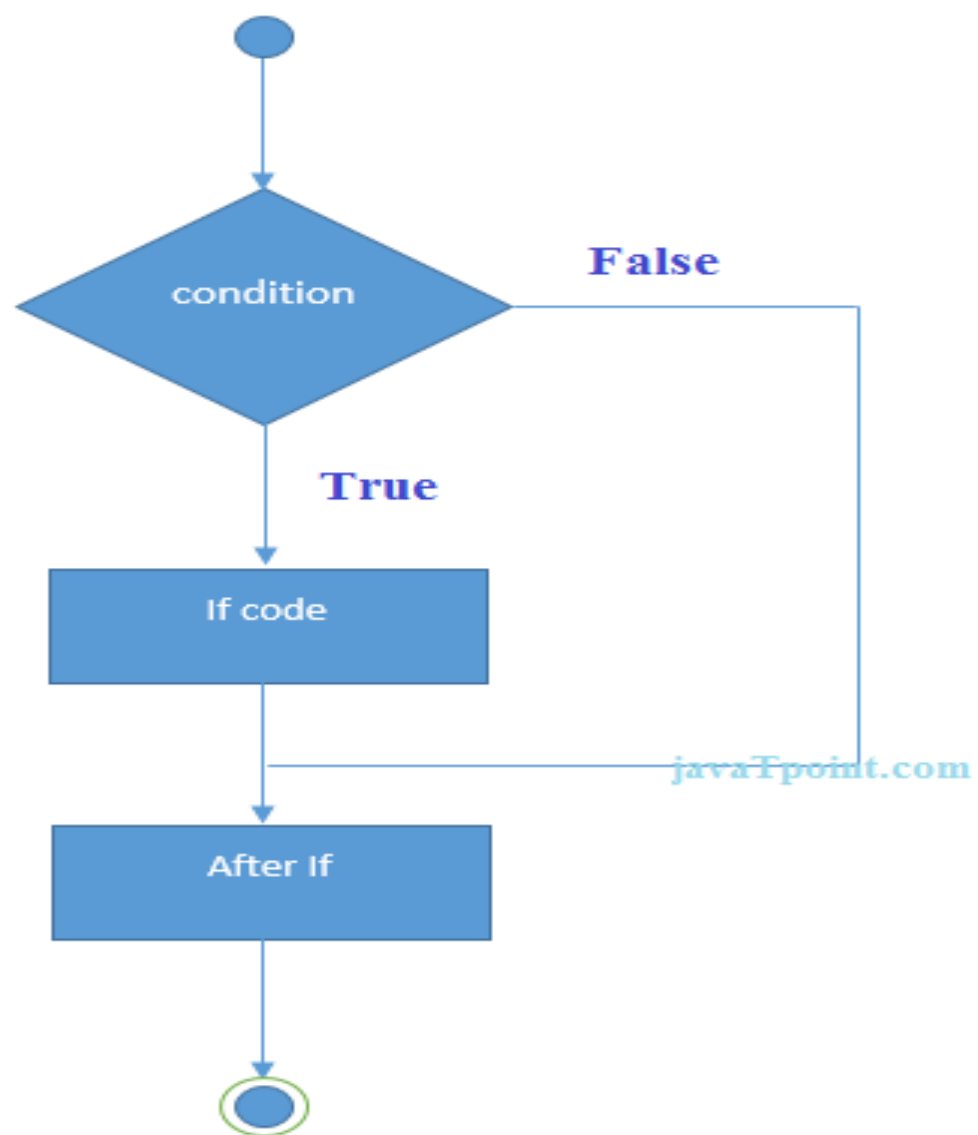
C if Statement

The syntax of the if statement in C programming is:

```
if (test expression)
{
    // code
}
```

The if statement evaluates the test expression inside the parenthesis ().

- If the test expression is evaluated to true, statements inside the body of if are executed.
- If the test expression is evaluated to false, statements inside the body of if are not executed.



C if...else Statement

The if statement may have an optional else block.

The syntax of the if..else statement is:

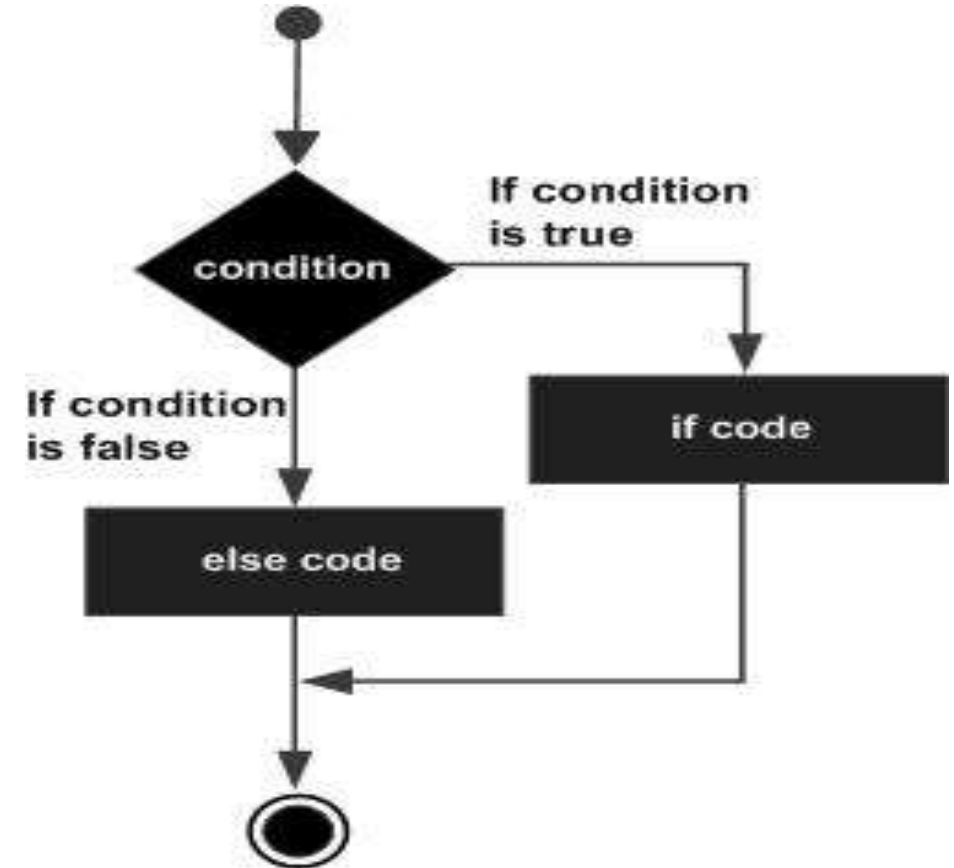
```
if (test expression) {  
    // run code if test expression is true  
}  
else {  
    // run code if test expression is false  
}
```


If the test expression is evaluated to true,

- statements inside the body of if are executed.
- statements inside the body of else are skipped from execution.

If the test expression is evaluated to false,

- statements inside the body of else are executed
- statements inside the body of if are skipped from execution.

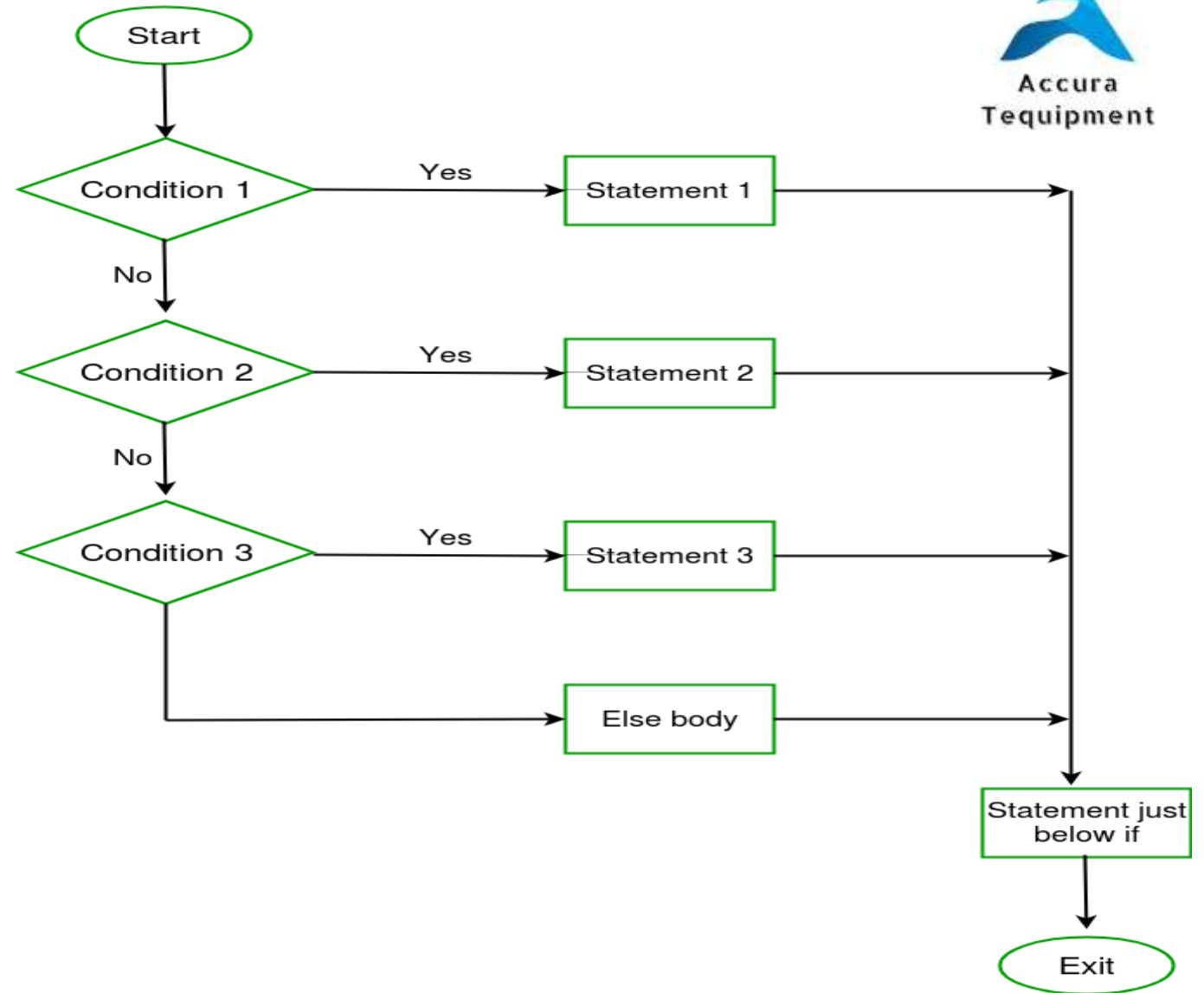


C if...else Ladder

- The if...else statement executes two different codes depending upon whether the test expression is true or false. Sometimes, a choice has to be made from more than 2 possibilities.
- The if...else ladder allows you to check between multiple test expressions and execute different statements.

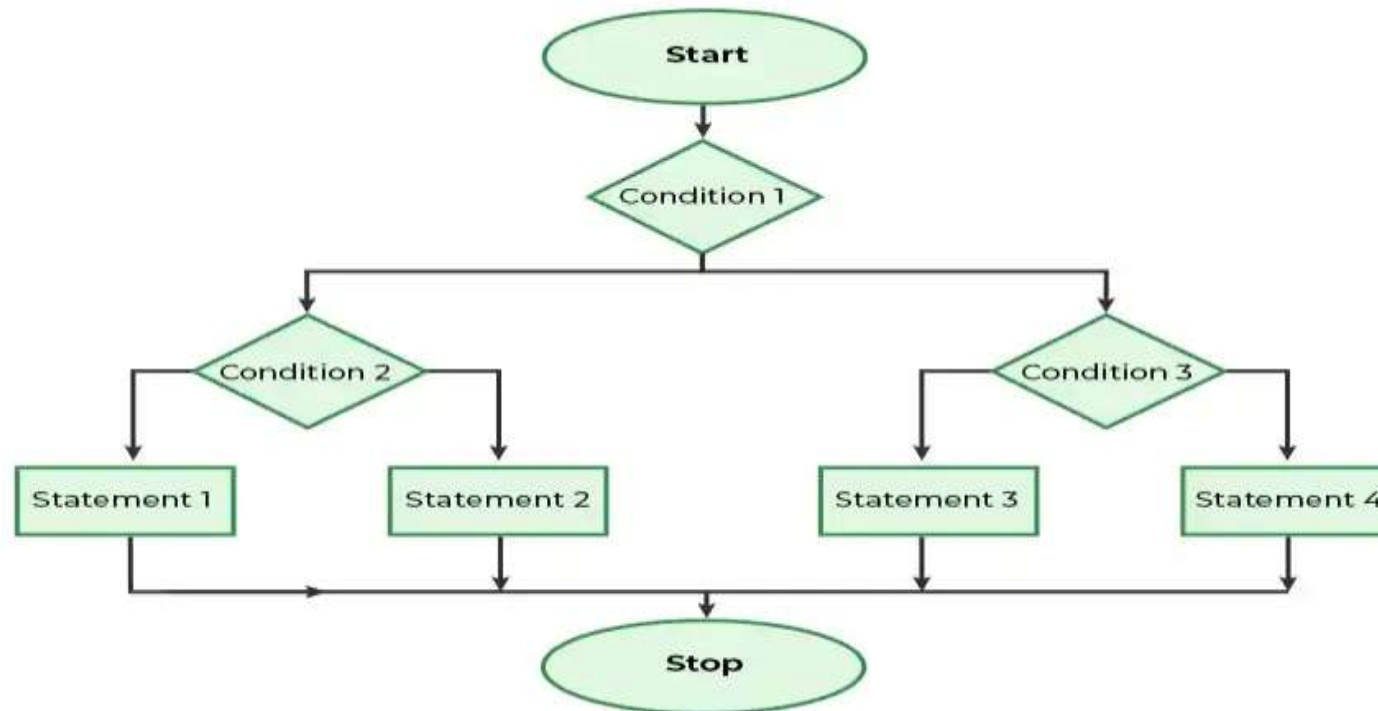
Syntax of if...else Ladder

```
if (test expression1) {  
    // statement(s)  
}  
else if(test expression2) {  
    // statement(s)  
}  
else if (test expression3) {  
    // statement(s)  
}  
.  
.  
else {  
    // statement(s)  
}
```



Nested if...else

- It is possible to include an if...else statement inside the body of another if...else statement.



C Loops

- In programming, a loop is used to repeat a block of code until the specified condition is met.

- C programming has three types of loops:
 - 1.for loop
 - 2.while loop
 - 3.do...while loop

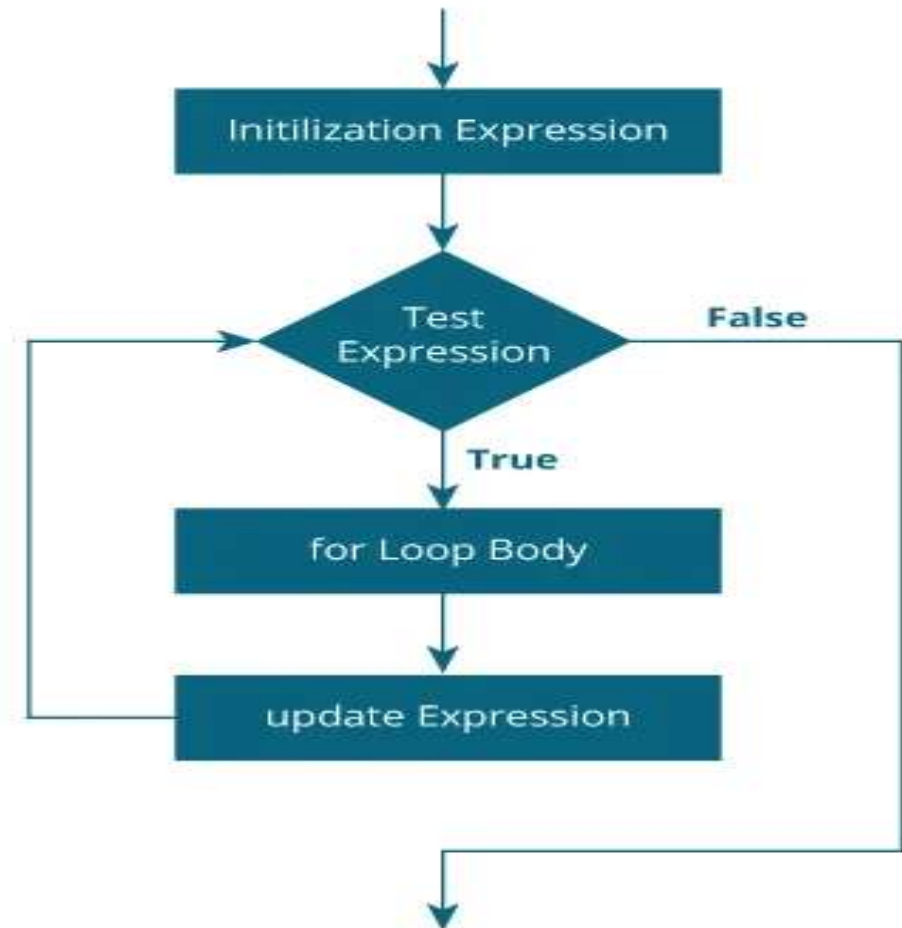
C for Loop

The syntax of the for loop is:

```
for (initializationStatement; testExpression; updateStatement)
{
    // statements inside the body of loop
}
```

Working of C for loop

- The initialization statement is executed only once.
- Then, the test expression is evaluated. If the test expression is evaluated to false, the for loop is terminated.
- However, if the test expression is evaluated to true, statements inside the body of the for loop are executed, and the update expression is updated.
- Again the test expression is evaluated.



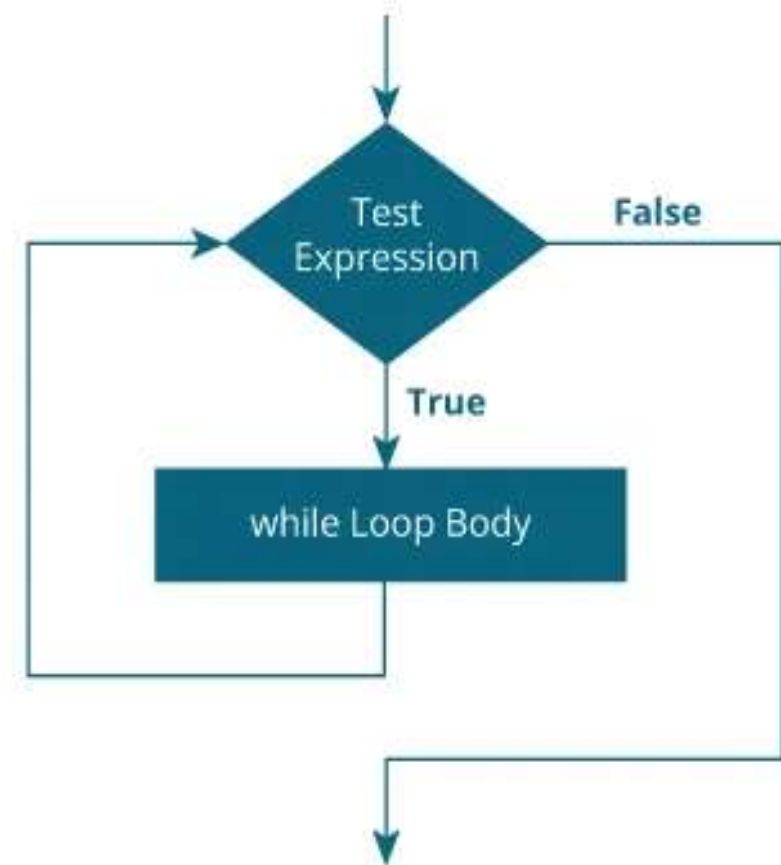
C while

The syntax of the while loop is:

```
while (testExpression) {  
    // the body of the loop  
}
```


Working of C while loop

- The while loop evaluates the testExpression inside the parentheses ().
- If testExpression is true, statements inside the body of while loop are executed. Then, testExpression is evaluated again.
- The process goes on until testExpression is evaluated to false.
- If testExpression is false, the loop terminates (ends).



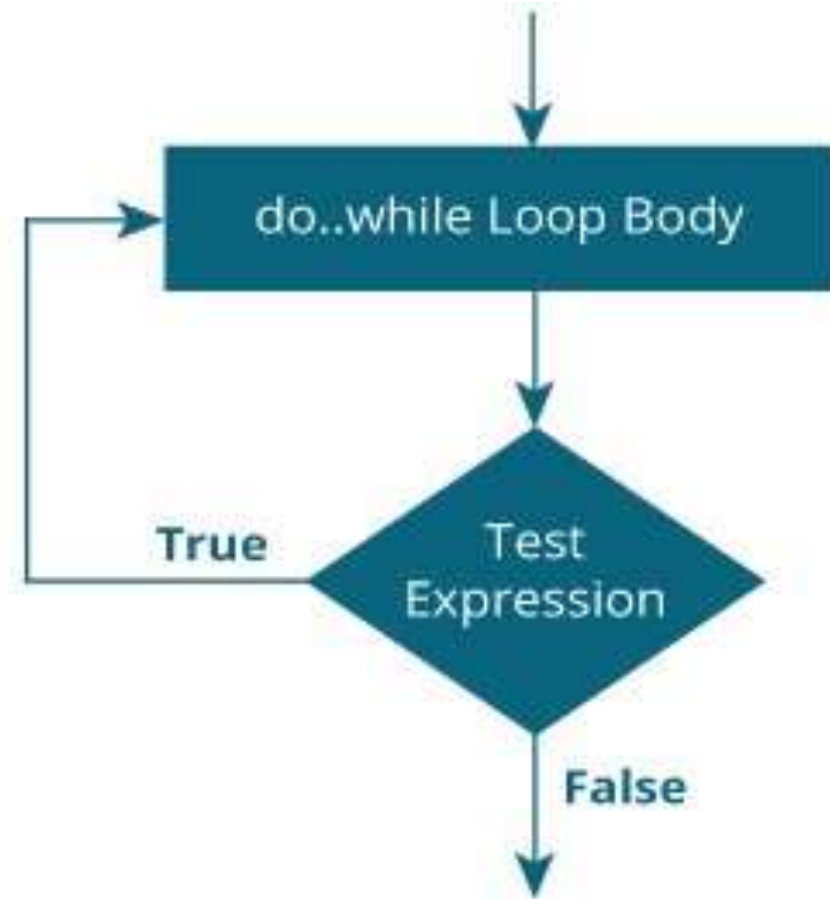
do...while loop

- The do..while loop is similar to the while loop with one important difference. The body of do...while loop is executed at least once. Only then, the test expression is evaluated.
- The syntax of the do...while loop is:

```
do {  
    // the body of the loop  
}  
while (testExpression);
```

Working of C do...while loop

- The body of do...while loop is executed once. Only then, the testExpression is evaluated.
- If testExpression is true, the body of the loop is executed again and testExpression is evaluated once more.
- This process goes on until testExpression becomes false.
- If testExpression is false, the loop ends.



C break and continue

C break

- The break statement ends the loop immediately when it is encountered. Its syntax is:

break;

- The break statement is almost always used with if...else statement inside the loop.

C continue

- The continue statement skips the current iteration of the loop and continues with the next iteration. Its syntax is:

continue;

- The continue statement is almost always used with the if...else statement.

C switch Statement

- The switch statement allows us to execute one code block among many alternatives.
- You can do the same thing with the if...else..if ladder. However, the syntax of the switch statement is much easier to read and write.

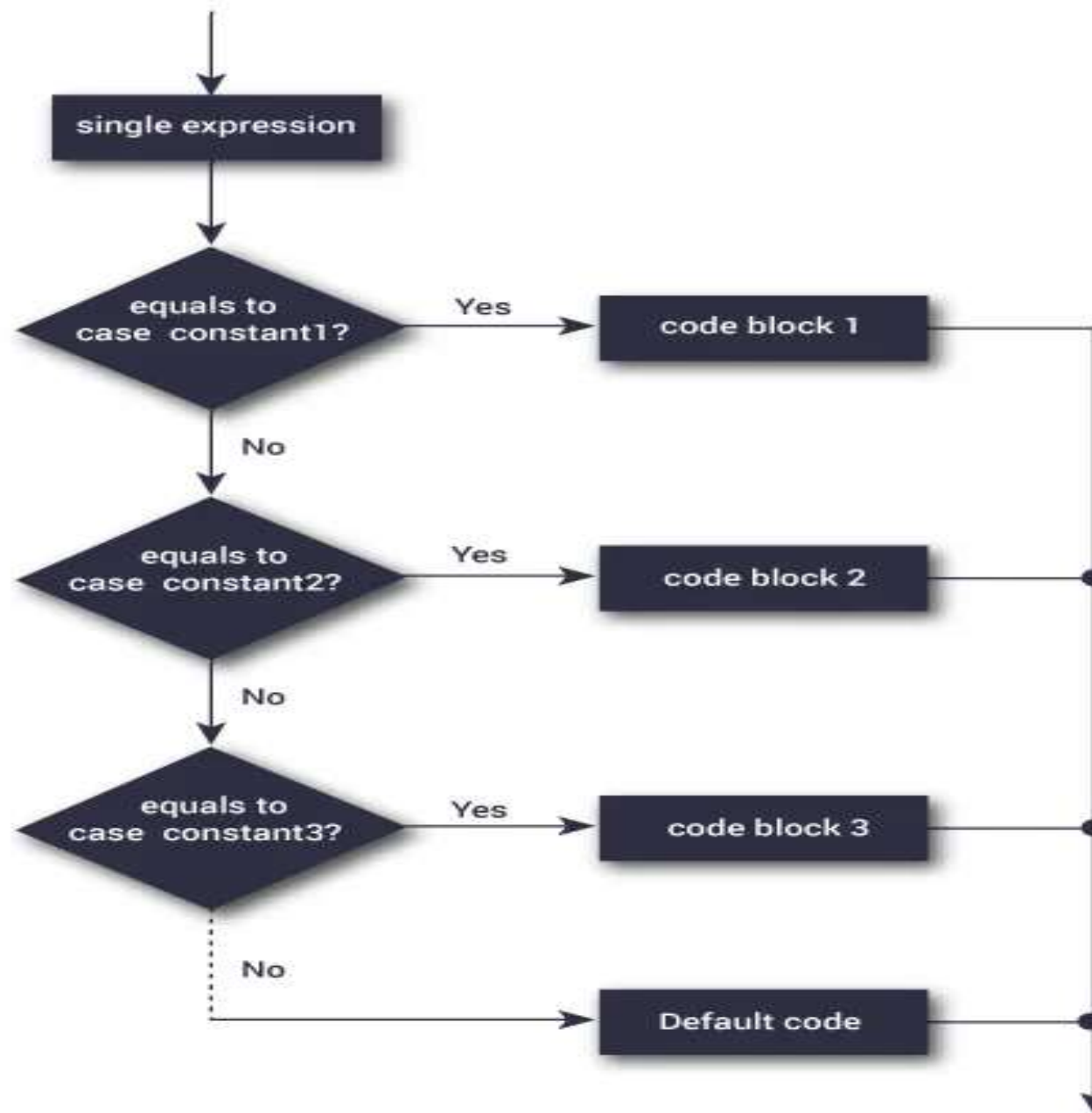
Syntax of switch...case

switch (expression)

```
{  
    case constant1:  
        // statements  
        break;  
    case constant2:  
        // statements  
        break;  
    .  
    .  
    .  
    default:    // default statements  
}
```

Working of switch statement

- The expression is evaluated once and compared with the values of each case label.
- If there is a match, the corresponding statements after the matching label are executed. For example, if the value of the expression is equal to constant2, statements after case constant2: are executed until break is encountered.
- If there is no match, the default statements are executed.



C goto Statement

- The goto statement allows us to transfer control of the program to the specified label.

Syntax of goto Statement

```
goto label;
```

```
... ..
```

```
... ..
```

```
label:
```

```
statement;
```

- The label is an identifier. When the goto statement is encountered, the control of the program jumps to label: and starts executing the code.

C Functions

- A function is a block of code that performs a specific task.
- Dividing a complex problem into smaller chunks makes our program easy to understand and reuse.

Types of function

- There are two types of function in C programming:
 1. Standard library functions
 2. User-defined functions

Standard library functions

- The standard library functions are built-in functions in C programming.
- These functions are defined in header files. For example,
 - The `printf()` is a standard library function to send formatted output to the screen (display output on the screen). This function is defined in the `stdio.h` header file. Hence, to use the `printf()` function, we need to include the `stdio.h` header file using `#include <stdio.h>`.
 - The `sqrt()` function calculates the square root of a number. The function is defined in the `math.h` header file.

User-defined function

- C allows you to define functions according to your need. These functions are known as user-defined functions.

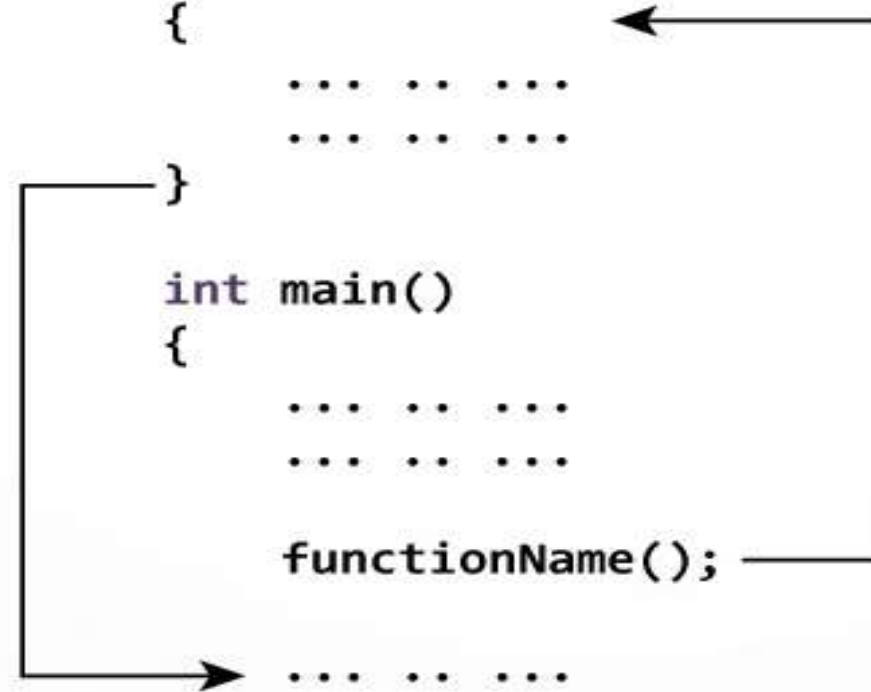
```
#include <stdio.h>
void functionName()
{
    ... ..
    ... ..
}
int main()
{
    ... ..
    ... ..
    functionName();
    ... ..
    ... ..
}
```

How function works in C programming?

```
#include <stdio.h>

void functionName()
{
    ... ..
    ... ..
}

int main()
{
    ... ..
    ... ..
    functionName();
    ... ..
    ... ..
}
```



Advantages of user-defined function

- 1.The program will be easier to understand, maintain and debug.
- 2.Reusable codes that can be used in other programs
- 3.A large program can be divided into smaller modules. Hence, a large project can be divided among many programmers.

Passing arguments to a function

- In programming, argument refers to the variable passed to the function
- The parameters a and b accepts the passed arguments in the function definition. These arguments are called formal parameters of the function.
- The type of arguments passed to a function and the formal parameters must match, otherwise, the compiler will throw an error.
- If n1 is of char type, a also should be of char type. If n2 is of float type, variable b also should be of float type.
- A function can also be called without passing an argument.

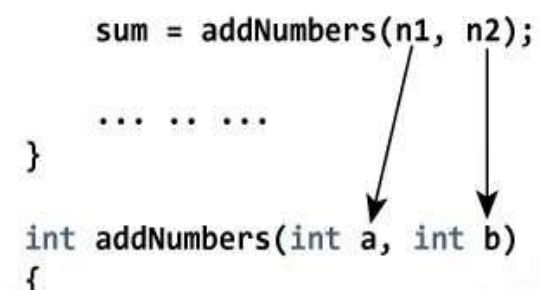
How to pass arguments to a function?

```
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... ..
    sum = addNumbers(n1, n2);
    ... ..
}

int addNumbers(int a, int b)
{
    ... ..
    ... ..
}
```



The diagram illustrates the flow of arguments from the `main` function to the `addNumbers` function. Two arrows originate from the arguments `n1` and `n2` in the function call `sum = addNumbers(n1, n2);` within the `main` function. These arrows point to the parameters `a` and `b` in the function definition `int addNumbers(int a, int b)`, demonstrating how the values are passed to the function's local variables.

Return Statement

- The return statement terminates the execution of a function and returns a value to the calling function. The program control is transferred to the calling function after the return statement.
- **Syntax of return statement**
return (expression);
- The type of value returned from the function and the return type specified in the function prototype and function definition must match.

Return statement of a Function

```
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... ..
    sum = addNumbers(n1, n2);
    ... ..
}

int addNumbers(int a, int b)
{
    ... ..
    return result;
}
```

sum = result

C Recursion

- A function that calls itself is known as a recursive function. And, this technique is known as recursion.
- The recursion continues until some condition is met to prevent it.
- To prevent infinite recursion, if...else statement (or similar approach) can be used where one branch makes the recursive call, and other doesn't.

```
void recurse()  
{  
    ... ..  
    recurse();  
    ... ..  
}  
  
int main()  
{  
    ... ..  
    recurse();  
    ... ..  
}
```


Advantages and Disadvantages of Recursion

- Recursion makes program elegant. However, if performance is vital, use loops instead as recursion is usually much slower.
- That being said, recursion is an important concept. It is frequently used in data structure and algorithms. For example, it is common to use recursion in problems such as tree traversal.

C Arrays

- An array is a variable that can store multiple values.
- An array is a group of related data items that share a common name.
- For example, if you want to store 100 integers, you can create an array for it. `int data[100];`

Declaration of an array:

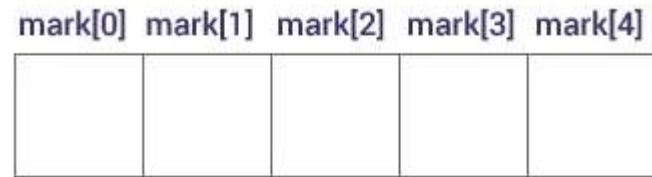
`dataType arrayName[arraySize];`

For example: `float mark[5];` Here, we declared an array, mark, of floating-point type. And its size is 5. Meaning, it can hold 5 floating-point values.

It's important to note that the size and type of an array cannot be changed once it is declared.

Access Array Elements

- To access elements of an array by indices.
- In the above example an array is declared as mark. The first element is mark[0], the second element is mark[1] and so on.



- Arrays have 0 as the first index, not 1. In this example, mark[0] is the first element.
- If the size of an array is n, to access the last element, the n-1 index is used. In this example, mark[4]

C Multidimensional Arrays

- In C programming, you can create an array of arrays. These arrays are known as multidimensional arrays.
- For example: `float x[3][4];`
- Here, `x` is a two-dimensional (2d) array. The array can hold 12 elements. You can think the array as a table with 3 rows and each row has 4 columns.

	Column 1	Column 2	Column 3	Column 4
Row 1	<code>x[0][0]</code>	<code>x[0][1]</code>	<code>x[0][2]</code>	<code>x[0][3]</code>
Row 2	<code>x[1][0]</code>	<code>x[1][1]</code>	<code>x[1][2]</code>	<code>x[1][3]</code>
Row 3	<code>x[2][0]</code>	<code>x[2][1]</code>	<code>x[2][2]</code>	<code>x[2][3]</code>

- Similarly, you can declare a three-dimensional (3d) array. For example: `float y[2][4][3];`
- Here, the array `y` can hold 24 elements.

C Structure

- In C programming, a struct (or structure) is a collection of variables (**can be of different types**) under a single name.

Define Structures

- Before you can create structure variables, you need to define its data type. To define a struct, the struct keyword is used.

Syntax of struct

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

For example,

```
struct Person {  
    char name[50];  
    int citNo;  
    float salary;  
};
```

Here, a derived type **struct Person** is defined. Now, you can create variables of this type.

```
#include <stdio.h>
#include <string.h>
struct Person {           // create struct with person1 variable
    char name[50];
    int citNo;
    float salary;
} person1;
int main() {              // assign value to name of person1
    strcpy(person1.name, "Asha Patil");
    // assign values to other person1 variables
    person1.citNo = 1984;
    person1.salary = 2500;
    printf("Name: %s\n", person1.name); // print struct variables
    printf("Citizenship No.: %d\n", person1.citNo);
    printf("Salary: %.2f", person1.salary);
    return 0;
}
```

Output



Name: Asha Patil

Citizenship No.: 1984

Salary: 2500.00

- In this program, we have created a struct named Person. We have also created a variable of Person named person1.

Need structs in C

- Suppose you want to store information about a person: his/her name, citizenship number, and salary. You can create different variables name, citNo and salary to store this information.
- If you need to store information of more than one person, then, you need to create different variables for each information per person: name1, citNo1, salary1, name2, citNo2, salary2, etc.
- A better approach would be to have a collection of all related information under a single name Person structure and use it for every person.

C Unions

- A union is a user-defined type similar to structs in C except for one key difference.
- Structures allocate enough space to store all their members, whereas **unions can only hold one member value at a time**.
- We use the **union** keyword to define unions. Here's an example:

```
union car  
{  
    char name[50];  
    int price;  
};
```

The above code defines a derived type union car.

Create union variables

- When a union is defined, it creates a user-defined type. However, no memory is allocated. To allocate memory for a given union type and work with it, we need to create variables.
- Here's how we create union variables.

```
union car
{
    char name[50];
    int price;
};
int main()
{
    union car car1, car2, *car3;
    return 0;
}
```

Another way of creating union variables is:

```
union car
{
    char name[50];
    int price;
} car1, car2, *car3;
```

In both cases, union variables car1, car2, and a union pointer car3 of union car type are created.

Access members of a union

- We use the `.` operator to access members of a union. And to access pointer variables, we use the `->` operator. In the above example,
- To access price for car1, car1.price is used.
- To access price using car3, either `(*car3).price` or `car3->price` can be used.

Difference between unions and structures

```
#include <stdio.h>
union unionJob
{
    //defining a union
    char name[32];
    float salary;
    int workerNo;
} uJob;

struct structJob
{
    char name[32];
    float salary;
    int workerNo;
} sJob;

int main()
{
    printf("size of union = %d bytes", sizeof(uJob));
    printf("\nsize of structure = %d bytes", sizeof(sJob));
    return 0;
}
```

Output

size of union = 32

size of structure = 40

- The difference in the size of union and structure variables is
- Here, the size of sJob is 40 bytes because
 - the size of name[32] is 32 bytes
 - the size of salary is 4 bytes
 - the size of workerNo is 4 bytes
- However, the size of uJob is 32 bytes. It's because the size of a union variable will always be the size of its largest element. In the above example, the size of its largest element, (name[32]), is 32 bytes.

