C Programming Short Notes

Part 2

1. Tokens in c?

- A token in C can be defined **as the smallest individual element of the C programming language that is meaningful to the compiler**. It is the basic component of a C program.

KEYWORDS

=> **Reserved words,** the meanings **are already been designed in the compiler**
=> 32 key words
=> **Case-sensitive**
=> Keywords cannot be used as constants or variables or any other identifiers name.

Keywords in C Language

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

IDENTIFIER

=> Identifier is a **name used to identify a variable, function, or any other user-defined item**

=> Same rules as variable name.

Rules of Identifier

=> Combination of alpha, number, and ('_') underscore.

=> Cannot start with number.

=> Case sensitive

=> Whitespace is not allowed in-between variables

eg:-

ab_c //valid
ab_12 //valid
_abc //valid
Auto //valid
abc12 //valid

1abc //invalid
abc#@ //invalid
a b //invalid

CONSTANT

=> **We can't change the fixed values constant during execution.**
=> The **fixed values** are known as **literals.**
=> Two types:- **Numeric constant, Character constant**

=> Numeric constant => real, octal, hexadecimal number
=> real number => decimal, floating-point

=> Character constant => single, multiple constant

Constant	Example
Decimal Constant	10, 20, 450 etc.
Real or Floating-point Constant	10.3, 20.2, 450.6 etc.
Octal Constant	021, 033, 046 etc.
Hexadecimal Constant	0x2a, 0x7b, 0xaa etc.
Character Constant	'a', 'b', 'x' etc.
String Constant	"c", "c program", "c in javatpoint" etc.

=> Two ways to declare constant

=> **const keyword**

```
const data_type variable_name = const_value;
```

=> **macro**

```
#define MACRO_NAME const_value
```

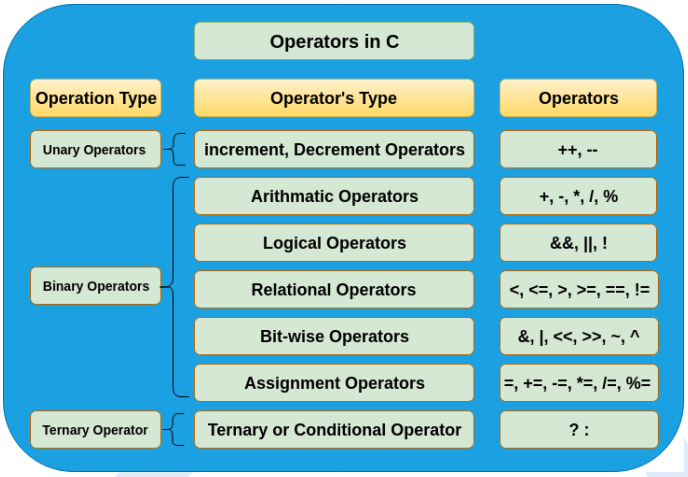
SPECIAL SYMBOLS

- **Brackets[]** − Opening and closing of brackets are used for array element reference, which indicate single and multidimensional subscripts.
- **Parentheses()** − These special symbols are used for function calls and function parameters
- **Braces{}** − Opening and closing of curly braces indicates the start and end of a block of code which contains more than one executable statement.
- **Comma (,)** − It is used to separate more than one statements like separating parameters in function calls.
- **Colon(:)** − It is an operator which essentially invokes something called as an initialization list.
- **Semicolon(;)** − It is called as a statement terminator that indicates the end of one logical entity. That is the reason each individual statement must be ended with a semicolon.
- **Asterisk (*)** − It is used to create a **pointer variable**.
- **Assignment operator(=)** − It is used for assigning values.
- **Pre-processor (#)** − The **pre-processor** called as a macro processor is used by the compiler to transform your program before the actual compilation starts.

Symbol	Name of symbol	symbol	Name of symbol
,	Comma	"	Double Quotation mark
;	Semi-colon	?	Question mark
(,)	Parentheses	:	Colon
[,]	Bracket	%	Percent
{,}	Braces or Curly bracket	~	Tilde
\	Back slash	+	Plus
'	Single quotation mark(right)	-	Minus
'	Single quotation mark(left)	_	Underscore
#	Hash	|	Pipe
@	At the rate of		
.	Decimal	!	Exclamation mark
<	Less than	/	Slash
>	Greater than	&	Ampersand
^	Carat	*	Asterisk

OPERATORS

An operator is a symbol that operates on a value or a variable



1. **Arithmetic Operators**:

   These operators perform mathematical operations on operands. (+, -, *, /, %, ++, --)

2. **Relational Operators:**

   These operators compare two operands and return true or false values based on the comparison. (<, >, <=, >=, ==, !=)

3. **Logical Operators:**

   These operators perform logical operations on boolean values (true or false). (&&, ||, !)

4. **Bitwise Operators:**

   These operators manipulate individual bits of operands. (&, |, ^, ~, <<, >>)

5. **Assignment Operators:**

   These operators assign values to variables. (+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=)

6. Conditional (Ternary) Operator (? :):

The conditional operator is a shorthand way to write an if-else statement.

Syntax: condition ? expression1 : expression2

If the condition is true, it evaluates to expression1; otherwise, it evaluates to expression2.

7. Comma Operator (,):

The comma operator evaluates multiple expressions sequentially and returns the value of the last expression.

It is often used in for loops or function calls.

2. Decision Making

- Decision-making in C involves using conditional statements to determine the flow of program execution based on certain conditions.
- C programming language assumes any non-zero and non-null values as true, and if it is either zero or null, then it is assumed as false value.

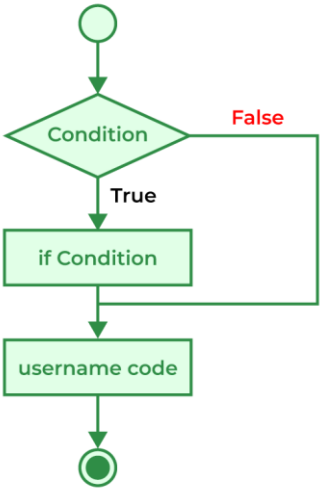
If Statement:

- The if statement is the most straightforward decision-making construct.
- It evaluates a condition and executes a block of statements if the condition is true.
- If the condition is false, the block is skipped.

Ex:-

```
#include <stdio.h>
int main() {
   int i = 10;
   if (i > 15) {
      printf("10 is greater than 15");
   }
   printf("I am Not in if");
}
```

Output: I am Not in if

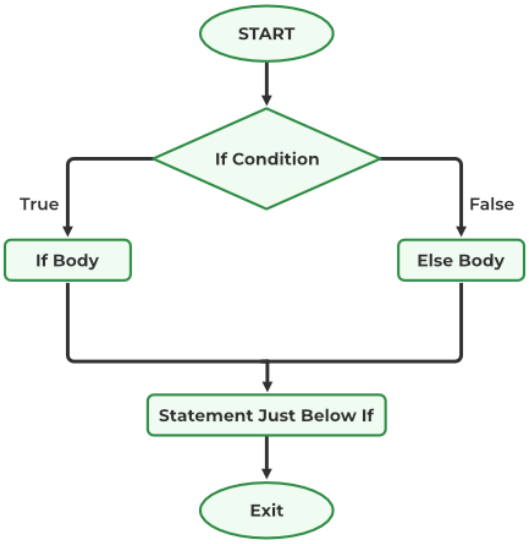


If-else Statement:

- The if-else statement allows us to execute different blocks of code based on whether a condition is true or false.
- If a certain condition is true then if the statement is executed otherwise else statement will be executed.

Syntax:

```
if (condition) {
   // Executes this block if condition is true
} else {
   // Executes this block if condition is false
}
```



Ex:-

```
#include <stdio.h>
int main() {
   int num = 5;
   if (num > 0) {
      printf("The number is positive");
   } else {
      printf("The number is non-positive");
   }
}
```

Output: The number is positive

Nested if Statement:

You can use nested if statements to create more complex decision structures.

Example:

```
#include <stdio.h>
int main() {
   int x = 10, y = 20;
   if (x == 10) {
      if (y == 20) {
         printf("Both x and y are 10 and 20, respectively");
      }
   }
}
```

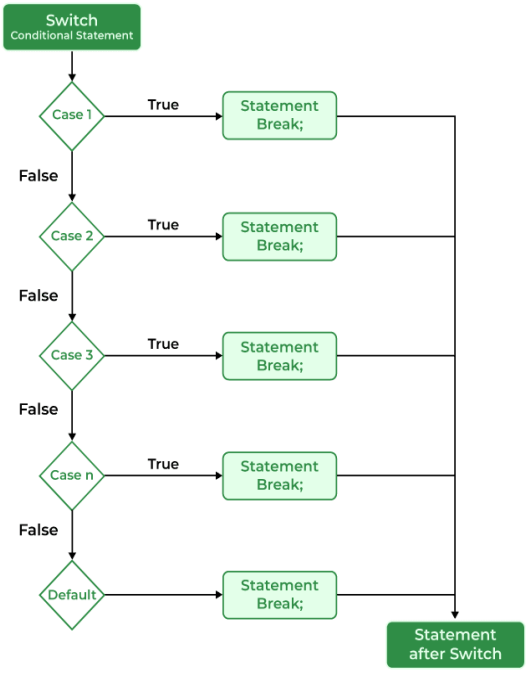
Output: Both x and y are 10 and 20, respectively

Switch Statement:

- The switch statement allows you to choose among several alternatives based on the value of an expression.
- It's particularly useful when you have multiple cases to handle.

Ex:-

```
#include <stdio.h>
int main() {
   char grade = 'A';
   switch (grade) {
      case 'A':
         printf("Excellent!");
         break;
      case 'B':
         printf("Good!");
         break;
      default:
         printf("Keep working hard!");
   }
}
// Output: Excellent!
```



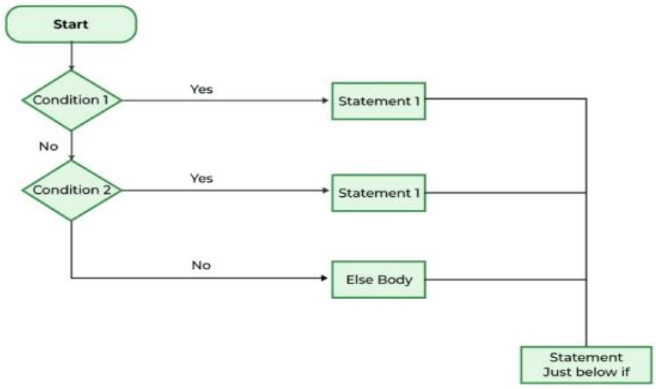
If ..else if …..else Statement:

- The else if construct allows you to handle multiple conditions sequentially.
- It's an extension of the if-else statement and is useful when you need to check additional conditions after the initial if condition.
- The else if construct allows you to test multiple conditions in a sequence.
- **Its executed from the top down. If one of the condition is true, the associated statement with the condition will execute and the rest of the statement will not execute.**

Example:

```
#include <stdio.h>
int main() {
   int num = 10;
```

```
    if (num > 0) {
        printf("The number is positive");
    } else if (num < 0) {
        printf("The number is negative");
    } else {
        printf("The number is zero");
    }
}
```



Output: The number is positive

3. Loops

- We need to execute a code multiple times. A loop statement is allowed to execute the block of code repeatedly until a certain condition is reached.

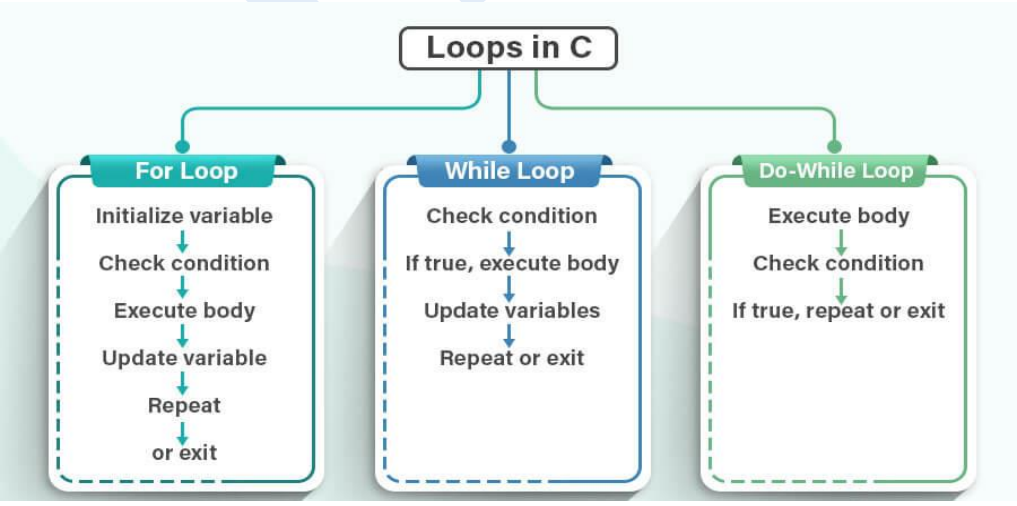
Types of Loops

Entry controlled loops : In Entry controlled loops the condition is checked before the entry of loop.

- **For loop**
- **While loop**

Exit controlled loops : In Exit controlled loops the condition is checked at the end of the loop. The loop will execute at least once whether the condition is false or true.

- **Do-while loop**

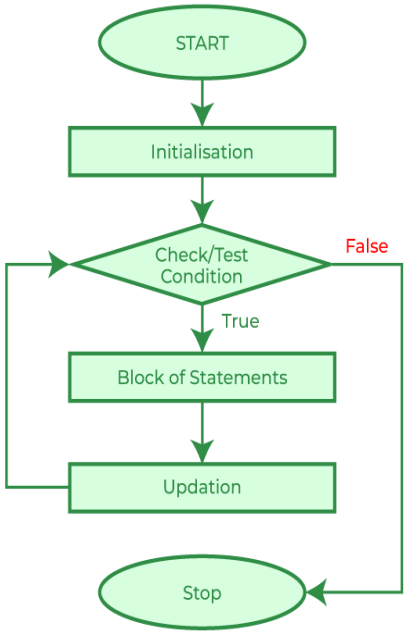


For loop

- A for loop is a repetition structure which allows us to **repeat a code of block for specific number of times.**

Syntax:

```
for(initialization; condition; updation)
{
//statement;
}
```



Example

```
#include <stdio.h>

int main() {
   printf("For loop:\n");
   for (int i = 0; i < 5; i++) {
      printf("%d ", i);
   }
   printf("\n");
   return 0;
}
```

Output

```
0 1 2 3 4
```

While loop

- While loop are used in situations where **we do not know the exact number of iteration of loop**. The loop execution is terminated based on the test condition.

Syntax:

```
Initialization;
while(test condition)
{
Statement;
Updation;
}
```

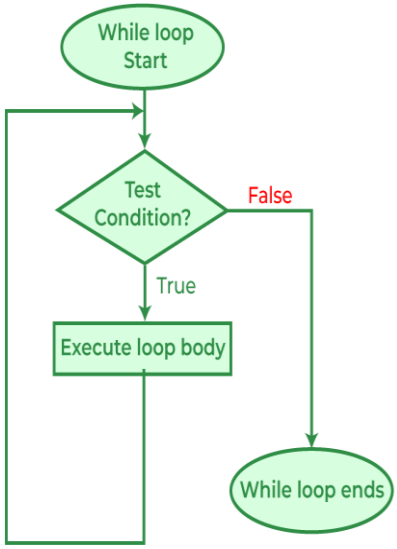
Example

```
#include <stdio.h>

int main() {
   printf("While loop:\n");
   int j = 0;
   while (j < 5) {
      printf("%d ", j);
      j++;
   }
   printf("\n");
   return 0;
}
```

Output

```
0 1 2 3 4
```

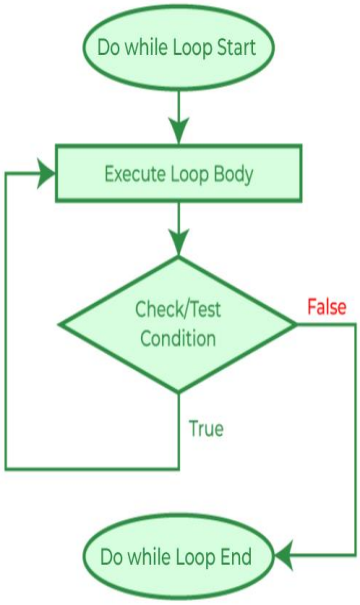


Do while Loop

- In do while loop **the condition is tested at the end of the loop.**
- The statement **execute at least once**

Syntax:

```
Initialization;
do
{
Statement;
Updation;
}while(test condition);
```



Example

```
#include <stdio.h>
int main() {
   printf("Do-while loop:\n");
   int k = 0;
   do {
      printf("%d ", k);
      k++;
   } while (k < 5);
   printf("\n");
   return 0;
}
```

Output

```
0 1 2 3 4
```

4. Jumping statements

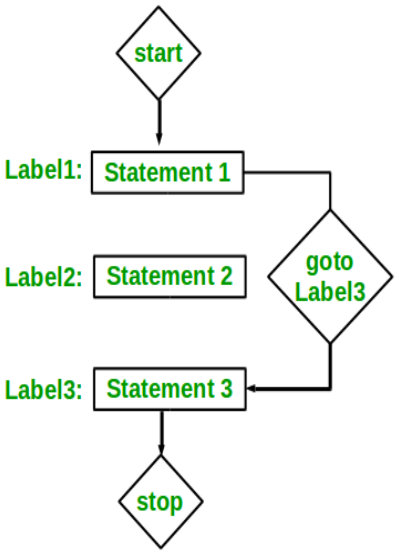
- These statements are used for **transferring control from one part of the program to another.**

1. goto Statement:

- The goto statement allows you to **jump to a labeled statement within the same function.**

Its syntax is:

goto label; // ... label: statement;



Example:

```
int i = 0;
loop:
   printf("%d", i);
   i++;
   if (i < 5)
      goto loop;
```

Output

0 1 2 3 4

2. break Statement:

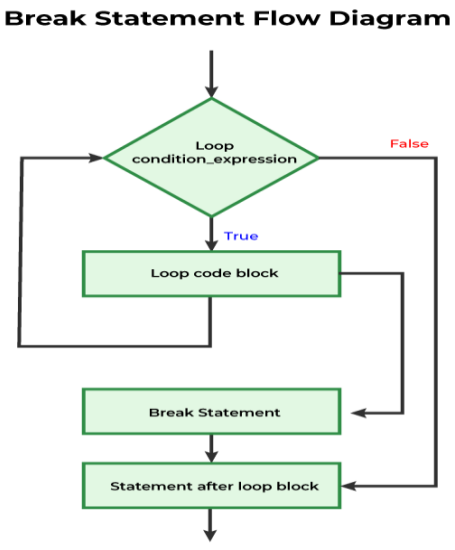
- The break statement is primarily used to exit from loops (for, while, do-while) and switch statements. It **immediately terminates the innermost loop or switch statement.**

Example:

```
for (int i = 0; i < 10; i++) {
   if (i == 5)
      break;
   printf("%d", i);
}
```

Output

0 1 2 3 4



3. continue Statement:

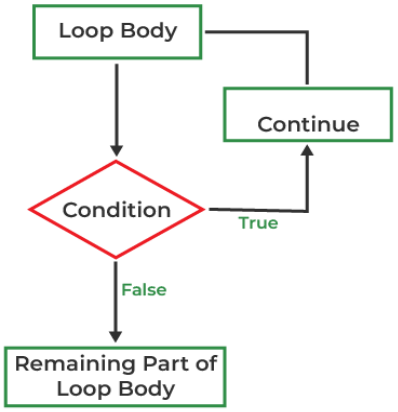
- The continue statement is used **within loops to skip the rest of the loop body and jump to the next iteration of the loop.**

Example:

```
for (int i = 0; i < 10; i++) {
   if (i == 5)
      continue;
   printf("%d\n", i);
   }
```

Output

```
0   1 2 3 4 6 7 8 9
```



4. exit Statement:

- The exit function is used to **terminate the program execution immediately.** It is part of the **stdlib.h** library. The exit function takes an integer argument, which is usually used to indicate the status of program termination.

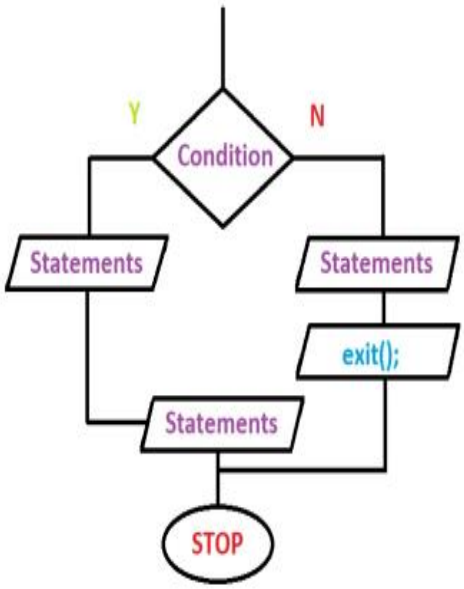
Example:

```
#include <stdlib.h>

int main() {
   // Some code...
   if (error_condition) {
      printf("An error occurred.\n");
      exit(EXIT_FAILURE);
   }
   // Rest of the code...
   return 0;
}
```

Output

```
An error occurred.
```



5. return Statement:

- The return statement is used to **terminate the execution of a function and return a value to the caller.** When a return statement is encountered within a function, **control is transferred back to the point where the function was called.**

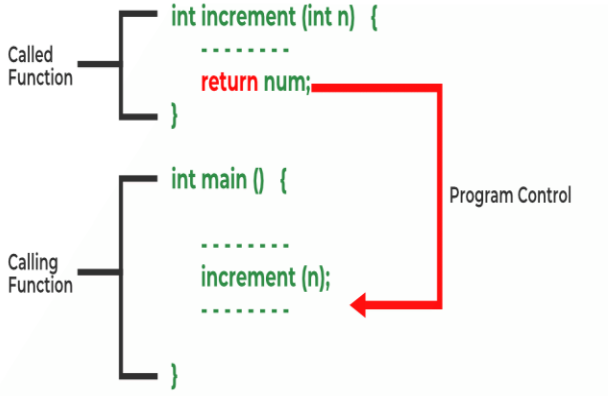
Example:

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

int main() {
    int sum = add(3, 5);
    printf("Sum: %d\n", sum);
    return 0;
}
```

Output

```
Sum: 8
```



5. Escape Sequence

- It is defined as a combination of \(backward slash) and letter or degit.
- Escape sequence are non-printable and are used to communicate with display device or printer by sending non graphical control

Escape Sequence	Meaning
\n	New Line
\t	Horizontal Tab
\b	BackSpace
\r	Carriage Return
\a	Audible bell
\'	Printing single quotation
\"	printing double quotation
\?	Question Mark Sequence
\\	Back Slash
\f	Form Feed
\v	Vertical Tab
\0	Null Value
\nnn	Print octal value
\xhh	Print Hexadecimal value

6. Debugging

- Debugging is a methodical process of finding and reducing the number off bugs, or defects in a computer program or a piece of electronic hardware thus making it behave as expected.

Bug in terms of softer

A software bug (or just "bug") is an error, flaw, mistake failure, or fault in a computer program that prevents it from behaving as intended producing an incorrect result.

What is debugger?

A debugger is a program which runs other programs in such a way as to let you see every step of program execution.

A debugger will let you stop the program while it is running change the program or program variables, and start the program running again.

In codeblocks

S1: Add breakpoint -> debug -> start/continue
S2: debug-> debugging window-> watches
S3: f7 to move the cursor.