Conceptual Session-2

Control Statements & Loop

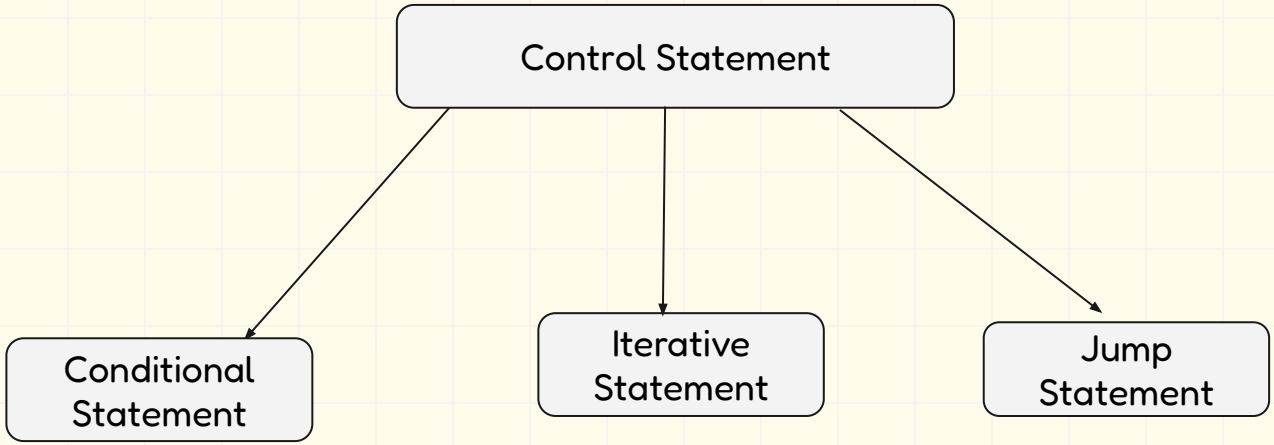
Presented by,
Shougoto Das
Senior CS Instructor
Phitron

Introduction to ASCII Table

The ASCII table is a character encoding standard that assigns unique numeric values to various characters. It stands for the **American Standard Code for Information Interchange**. The ASCII table consists of 128 characters, including letters, digits, special symbols, and control characters. These characters are represented by their corresponding ASCII values, which are decimal numbers ranging from 0 to 127.

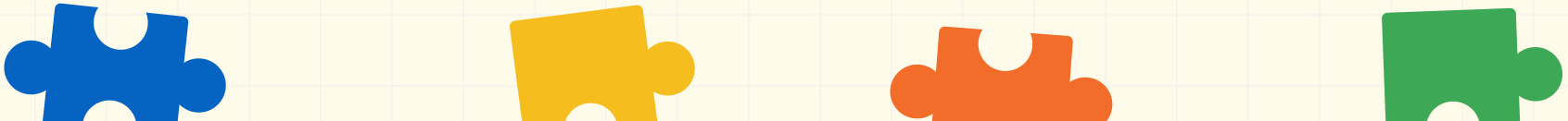
```
cook@pop-os:~$ ascii -d
    0 NUL     16 DLE     32          48 0     64 @     80 P     96 `    112 p
    1 SOH     17 DC1     33 !        49 1     65 A     81 Q     97 a    113 q
    2 STX     18 DC2     34 "        50 2     66 B     82 R     98 b    114 r
    3 ETX     19 DC3     35 #        51 3     67 C     83 S     99 c    115 s
    4 EOT     20 DC4     36 $        52 4     68 D     84 T    100 d    116 t
    5 ENQ     21 NAK     37 %        53 5     69 E     85 U    101 e    117 u
    6 ACK     22 SYN     38 &        54 6     70 F     86 V    102 f    118 v
    7 BEL     23 ETB     39 '        55 7     71 G     87 W    103 g    119 w
    8 BS      24 CAN     40 (        56 8     72 H     88 X    104 h    120 x
    9 HT      25 EM      41 )        57 9     73 I     89 Y    105 i    121 y
   10 LF      26 SUB     42 *        58 :     74 J     90 Z    106 j    122 z
   11 VT      27 ESC     43 +        59 ;     75 K     91 [    107 k    123 {
   12 FF      28 FS      44 ,        60 <     76 L     92 \    108 l    124 |
   13 CR      29 GS      45 -        61 =     77 M     93 ]    109 m    125 }
   14 SO      30 RS      46 .        62 >     78 N     94 ^    110 n    126 ~
   15 SI      31 US      47 /        63 ?     79 O     95 _    111 o    127 DEL
```

Understanding Control Statements in C



Understanding Control Statements in C

Loop statements (for, while, do-while): **Loop statements** in C allow us to execute a block of code repeatedly until a certain condition is met. They provide a way to perform iterative tasks efficiently. The three main types of loop statements in C are the **for loop**, **while loop**, and **do-while loop**. The **for loop** allows us to repeatedly execute a block of code for a fixed number of times. It consists of an initialization statement, a condition, and an update statement. The **while loop** executes a block of code as long as a given condition is true. The condition is evaluated before each iteration. The **do-while loop** is similar to the while loop, but it guarantees that the block of code is executed at least once, as the condition is evaluated after each iteration. We will now explore the syntax and usage of these loop statements in C, providing examples to illustrate their functionality.



Exploring Loop Statements in C

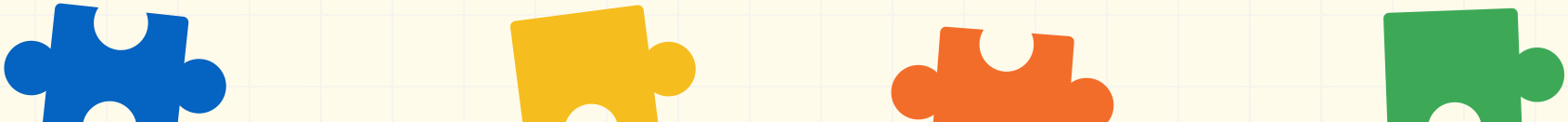
Basic loop structure in C: In C, the **for loop** provides a concise way to execute a block of code for a fixed number of times. Its structure consists of an initialization statement, a condition, and an update statement. The condition is evaluated before each iteration, and if it is true, the loop body is executed. After each iteration, the update statement is executed. The basic syntax of a for loop in C is as follows:

```
for (initialization; condition; update) {
    // code to be executed
}
```

Let's consider an example to understand the usage of the for loop.

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

This loop will execute the code block five times, printing the iteration number each time.



Exploring While Loop Statements in C

While loop in C,

initialization

while (condition) {

    // code to be executed while condition is true

increment/decrement

}

```
int i = 0;

while(i<5)

{

        printf("Iteration: %d\n", i);

        i++;

}
```




Exploring do while loop in C

Do while loop in C,

initialization

do {

// code to be executed

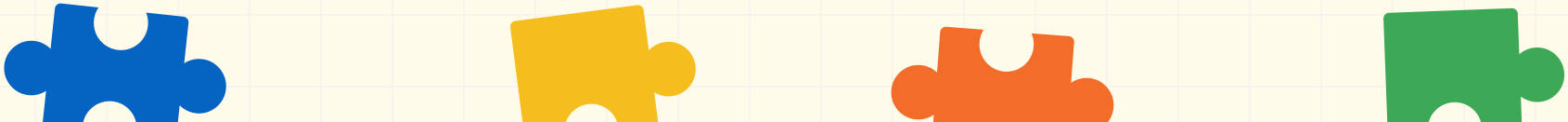
increment/decrement

} while (condition);

```
int i = 0;

do

{

        printf("Iteration: %d\n", i);

        i++;

}while(i<5);
```

Utilizing Nested Loops in C

Nested loop concept and structure: **Nested loops** in C refer to the situation where one loop is present inside another loop. They allow us to perform repetitive tasks with more complex patterns and structures. By nesting loops, we can iterate over multiple dimensions and perform operations on multidimensional data structures. The structure of nested loops consists of an outer loop and an inner loop. The outer loop controls the execution of the inner loop. For each iteration of the outer loop, the inner loop is executed completely. Nested loops provide a powerful tool for solving problems involving patterns, matrices, and complex data structures. Let's explore the concept and structure of nested loops in C with examples to illustrate their usage.



Utilizing Nested Loops in C

Nested loop examples and applications: Nested loops find applications in various programming scenarios. They are particularly useful when dealing with multidimensional arrays, generating patterns, and performing iterative calculations. Let's consider an example where we use nested loops to print a pattern of asterisks:

```
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < i; j++) {
        printf("* ");
    }
    printf("\n");
}
```

This code will generate the following pattern:

```
*
* *
* * *
* * * *
```

Nested loops allow us to iterate over each row and column of the pattern, controlling the number of asterisks printed in each row.




Problems

https://codeforces.com/group/MWSDmqGsZm/contest/219432/problem/B

https://codeforces.com/group/MWSDmqGsZm/contest/219432/problem/C

Problem 3:

For N = 5,Print the pattern,

*****

****

***

**

*