C Programming Lecture 7

Saturday, 15 June 2024 8:16 PM

Control Structures in C

These structures allow you to control the execution of your code based on certain conditions or iterate through a set of statements multiple times.

Loops, such as the for loop, while loop, and do-while loop, allow you to repeat a block of code multiple times until a specified condition is met. Conditionals, like the if statement and else statement, enable you to selectively execute certain blocks of code based on specific conditions. Switches, on the other hand, provide a way to perform different actions based on the value of a variable.

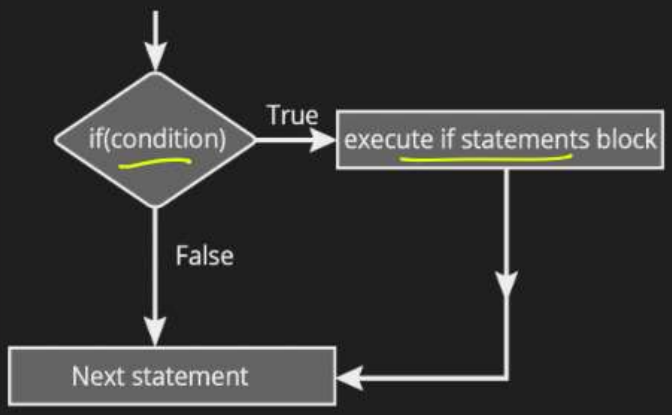
Selection Statements

The selection statements in C is used to perform the operations based on some specific condition. The operations specified in if block are executed if and only if the given condition is true.

There are the following variants of if statement in C language.

- If statement
- If-else statement
- If else-if ladder
- Nested if

if statement

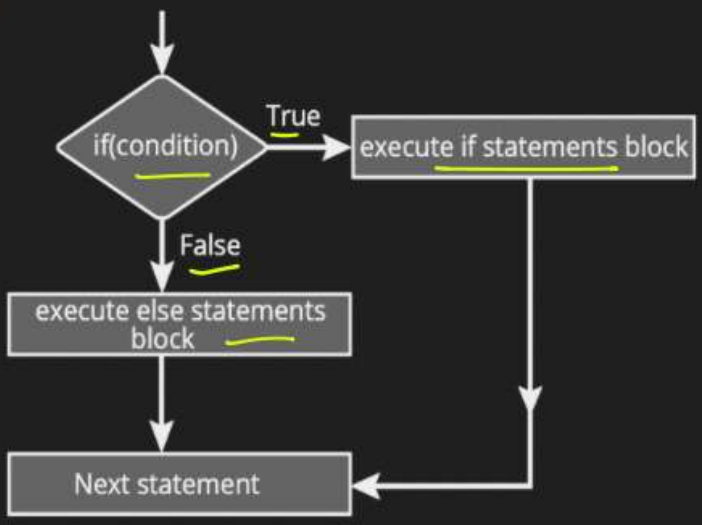


even

number % 2 == 0

```c
#include<stdio.h>
int main(){
    int number = 10;
    if(number%2 == 0){     // Can ignore the block {} if only one statement
        printf("%d is even number",number);
    }
    return 0;
}
```

if-else Statement




```c
if (num%2 == 0)
    printf("even");
else
    printf("odd");
```

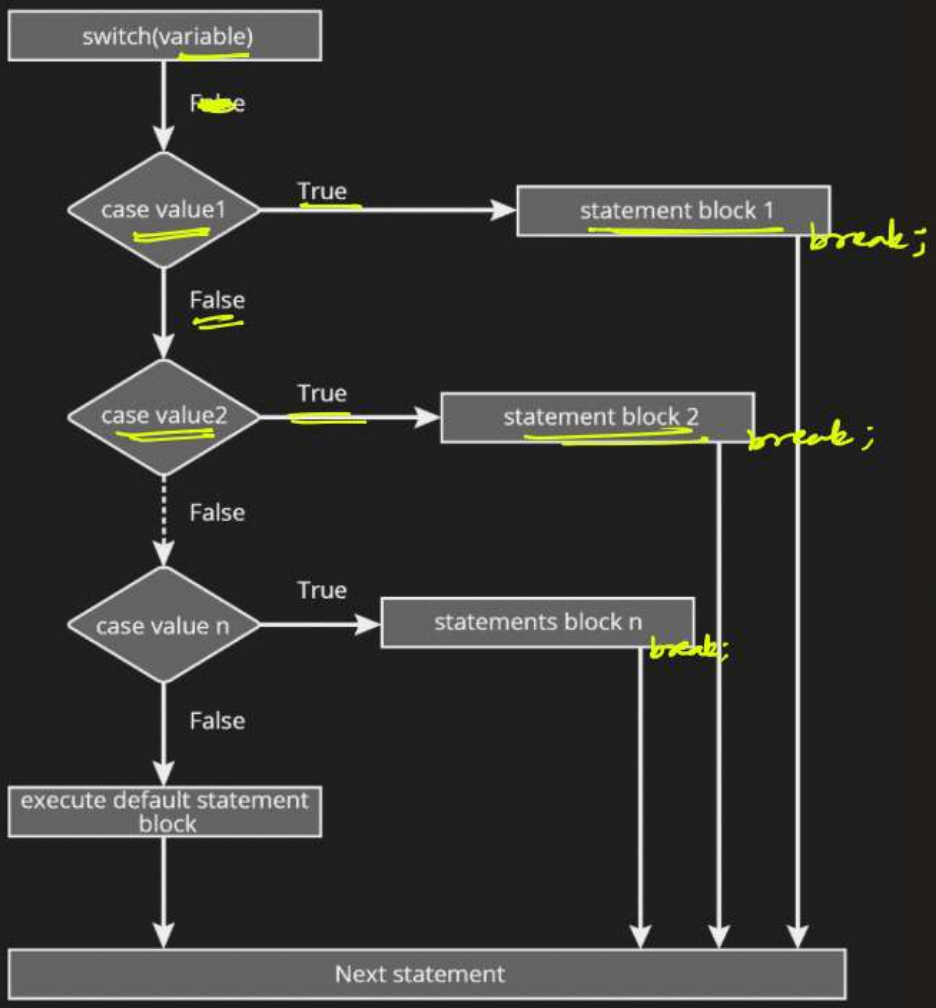
```c
#include<stdio.h>
int main(){
    int number = 13;
    if(number%2 == 0) {
        printf("%d is even number",number);
    }
    else{
        printf("%d is odd number",number);
    }
    return 0;
}
```

if else-if else statement

```c
#include<stdio.h>
int main() {
    int a = 10,
        b = 23,
        c = 7;
    if(a>=b && a>=c) {
        printf("%d is largest",a);
    }
    else if(b>=a  && b>=c) {
        printf("%d is largest",b);
    }
    else if(c>=a && c>=b) {
        printf("%d is largest",c);
    }
    return 0;
}
```

```c
num = 10
if (num%2==0) {
    printf("even");
}
else if (num%5==0){
    printf("div by 5");
}
else {
    ..
}
```

switch-case statement



case value1:

1. The *switch expression* must be of an integer or character type.
2. The *case value* must be a constant.
3. The *case value* can be used only inside the switch statement.
4. The *break statement* in switch case is not must. It is optional. If there is no break statement found in the case, all the cases will be executed present after the matched case. It is known as *fall through* the state of C switch statement.

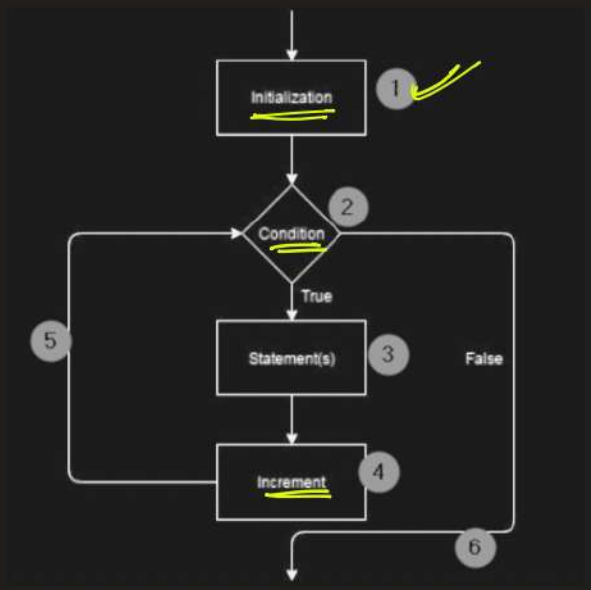
```c
#include<stdio.h>

int main(){
    int num = 2;
    switch(num) {
        case 1:
            printf("Value is 1\n");
            break;
        case 2:
            printf("Value is 2\n");
            break;
        case 3:
            printf("Value is 3\n");
            break;
        default:
            printf("Value is not 1, 2, or 3\n");
    }
    return 0;
}
```

Repetition Statements / Loops

The looping can be defined as repeating the same process multiple times until a specific condition satisfies. There are three types of loops used in the C language.

for loops



until a condition is satisfied

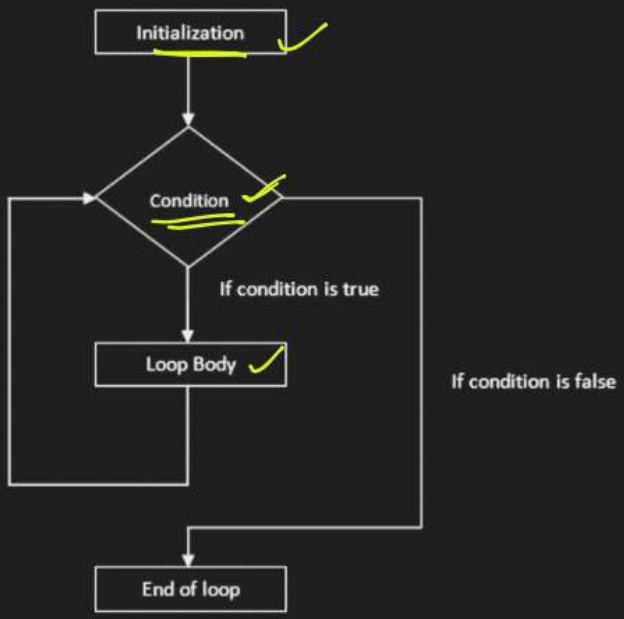
for(initialization; condition; inc/dec) {
statements
}

```c
#include<stdio.h>
int main(){
    for(int i=1; i<=10; i++){
        printf("%d \n",i);
    }
    return 0;
}
```

i 1 2 3 ... 9 10
11

```c
#include<stdio.h>
int main(){
    int i = 1;
    for( ; ; ){
        printf("%d \n",i);
        i++;
        if(i>10)
            break;
    }
    return 0;
}
```

while loops




```
int x = 0;
while (condition) {
    statements
    inc/de;
}
```

for while

End of loop

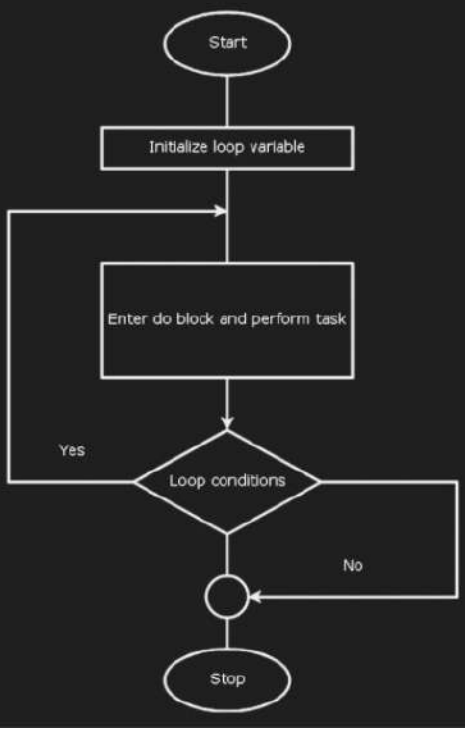
```c
#include<stdio.h>
int main(){
    int i = 1;
    while(i <= 10){
        printf("%d \n",i);
        i++;
    }
    return 0;
}
```

i 1 2 ... 10

for while

Condition is checked first then we go to statement

do-while loops



do {
{ }
} while ( )

Exit controlled

```c
int correct_pswd = 327, pswd;
do {
{ scanf ("Enter your pswd : %d", &pswd);
} while( pswd != correct_pswd);
```

```c
#include<stdio.h>
int main(){
    int i = 1;
    do {
        printf("%d\n", i);
        i++;
    } while(i <= 10);
    return 0;
}
```

Nested loops

```c
#include<stdio.h>
int main(){
   for(int i=1; i<=10; i++) {   // Tables of 1-10
       printf("Table of %2d: ", i);
       for(int j=1; j<=10; j++)   // inner loop
           printf("%3d ", i*j);
       printf("\n");
   }
   return 0;
}
```

continue statement

The continue statement in a loop causes the current iteration to end immediately, skipping any remaining code in the loop body for that iteration. It then proceeds to the next iteration of the loop.

```c
#include<stdio.h>
int main(){
    // Loop from 1 to 10
    for(int i = 1; i <= 10; i++) {
        // If the number is even, skip the rest of the loop body
        if(i%2 == 0)
            continue;

        // This will only print if the number is odd
        printf("%d ", i);
    }
    printf("\n"); // New line for better formatting
    return 0;
}
```

break; continue. } → nearest loops

1 3 5 7 9

continue statement in nested loops

```c
#include<stdio.h>
int main(){
    int numberOfSets, numberOfNumbers, num, sum;

    // Ask user for the number of sets
    scanf("%d", &numberOfSets);

    // Loop through each set
    for(int i = 0; i < numberOfSets; i++) {
        // Ask user for the number of numbers in the current set
        scanf("%d", &numberOfNumbers);

        sum = 0;

        // Loop through each number in the current set
        for(int j = 0; j < numberOfNumbers; j++) {
            scanf("%d", &num);

            // If the number is negative, skip the rest of the inner loop body
            if(num < 0)
                continue;

            sum += num;
        }
        // Print the sum of positive numbers
            printf("Sum of Positive numbers: %d\n", sum);
    }
    return 0;
}
```

t test cases ← input

{ n integers ← input

{ ------ ← input

positive

0 → n

400,000,000 B

400 MB

1000 test cases

TC1

{ 10^5 integers.

TC2

{ 10^5 int —

O(n)

10^8

break statement in nested loops

In C (and many other programming languages), the break statement is used to immediately terminate the nearest enclosing loop (e.g., for, while, do-while) or switch statement. It transfers control to the statement following the terminated loop or switch block, effectively exiting the loop prematurely based on a condition or requirement.

```c
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

int main(){
    srand(time(NULL)); // Initialize random seed based on current time
     while(1) {
        // Generate a random number between -10 and 10
        int num = rand() % 21 - 10; // Generates numbers from -10 to 10

        // Print the generated number
        printf("Generated number: %d\n", num);

        // If the number is negative, break out of the loop
        if(num < 0) {
            printf("Negative number encountered, stopping the loop.\n");
            break;
        }
    }
    return 0;
}
```

goto statement

The goto statement in C allows transferring control to another part of the program. It is often considered harmful due to its potential to create spaghetti code and make programs harder to understand and maintain.

Rules and Considerations:

- A label in C is a valid identifier followed by a colon (:). For example, start: is a label.
- Syntax:
  - goto label;
    Here, label is the identifier of a label defined elsewhere in the current function.
- Jumping Control:
  - When goto label; is executed, control jumps to the statement immediately following **label:** in the program.
- Scope:
  - Labels and goto statements are only effective within the same function scope. They cannot jump between different functions.
- Restrictions:
  - goto cannot be used to jump into a loop or switch statement from outside its body. It can only jump within the same function and cannot jump out of or into a different scope (like loops, switch blocks, or functions).
- Use Cases:
  - goto can sometimes be useful for breaking out of deeply nested loops or handling error conditions where multiple cleanup steps are needed.
- Best Practices:
  - Avoid overuse: Structured control flow statements (for, while, do-while, switch) are generally more readable and maintainable.
  - Limit use: If used, goto should be used sparingly and with caution, ensuring that it improves code clarity rather than complicating it.

```c
#include<stdio.h>
int main(){
    int i = 1;
    start:
    printf("Value of i: %d\n", i);
    i++;
    if(i <= 5) {
        goto start;
    }
    printf("End of program\n");
    return 0;
}
```

label for goto statement

1
2
3
4
5
EOP

[HW] Example 1: Find the Sum of Natural Numbers (1-n)

[HW] Example 2: Generate Fibonacci Series (First n fibonacci numbers)

[HW] Example 3: Check if a number if prime

Example 4: Check if a number is palindrome

```c
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int main(){
    int n = 123241;
    int rn = 0, on = n; // reversed_number, original_number
    while(n) {
        rn = rn*10+ n%10;
        n /= 10;
    }
    if(rn == on) {
        printf("Palindrome!!!");
    }
    return 0;
}
```