Short-term Hands-on Supplementary Course on C Programming

Session 3: Looping Statements

Nov 23, 2022

1 Loop Flow of Control

In programming, sometimes there is a need to perform some operation more than once or (say) n number of times. Loops come into use when we need to **repeatedly execute a block of statements**. There are primarily three types of looping constructs in C:

- for loop
- while-do loop
- do-while loop

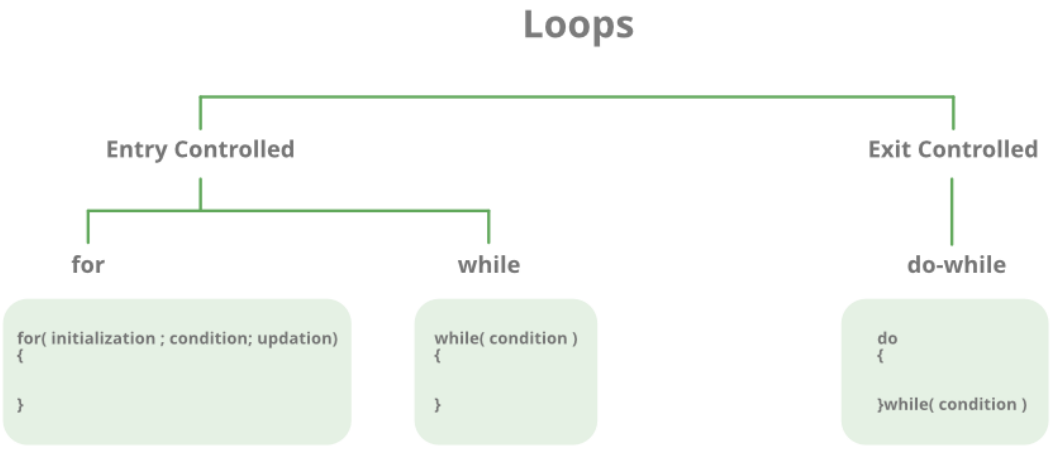

Figure 1: The different types of looping constructs in C.

There are mainly two types of control for loops:

1. **Entry Controlled loops:** In this type of loop, the test condition is tested before entering the loop body. For Loop and While Loop is entry- controlled loops.
2. **Exit Controlled Loops:** In this type of loop the test condition is tested or evaluated at the end of the loop body. Therefore, the loop body will execute at least once, irrespective of whether the test condition is true or false. the do-while loop is exit controlled loop

.

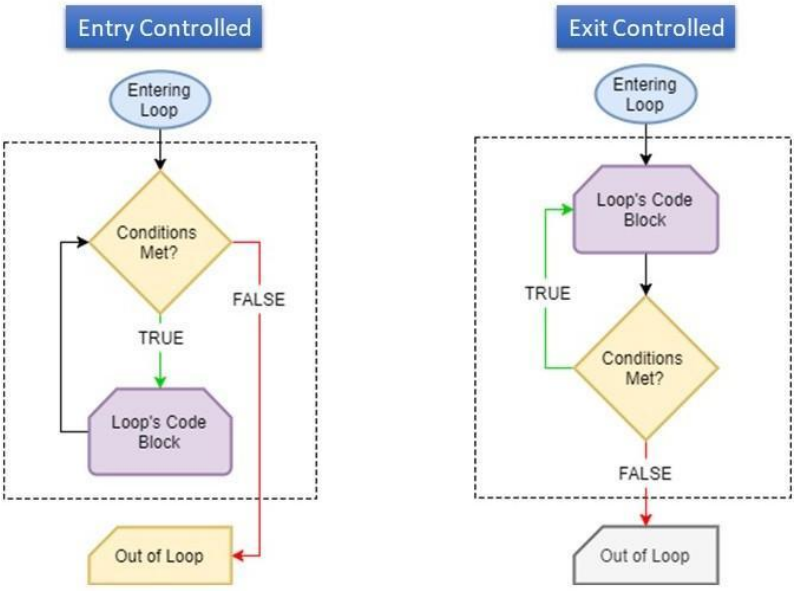

Figure 2: The difference between the execution of entry and exit controlled looping constructs.

1.1 for-loop

```
// for-loop
for (init; test; post-proc) {
   body ;
}
```

```
// Sum from 1 to 1000
int sum = 0, number;
for (number = 1; number <= 1000; ++number) {
   sum += number;
}
```

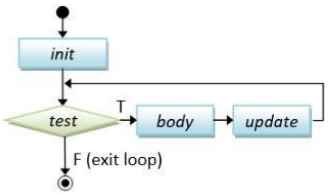

Figure 3: Syntax, example and flowchart of the for looping construct.

```c
#include <stdio.h>

int main(void) {

  int n;
  int sum = 0;

  printf("Enter the value of n: ");
  scanf("%d", &n);

  for(int i=1; i<=n; i++)
    {
      sum += i;
    }

  printf("The sum of %d n numbers is %d\n", n, sum);

    return 0;
}
```

Listing 1: Code to find the sum of first n non-zero numbers using for-loop in C.

1.2 while-do loop

```
// while-do
while ( condition ) {
   body ;
}
```

```
int sum = 0, number = 1;
while (number <= 1000) {
   sum += number;
   ++number;
}
```

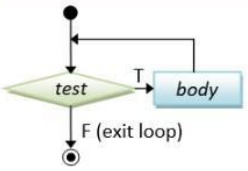

Figure 4: Syntax, example and flowchart of the while-do looping construct.

```c
#include <stdio.h>

int main(void) {

  int n;
  int sum = 0;

  printf("Enter the value of n: ");
  scanf("%d", &n);

  int i = 1;
  while(i <= n)
    {
      sum += i++;
    }

  printf("The sum of %d numbers is %d\n", n, sum);

    return 0;
}
```

Listing 2: Code to find the sum of first n non-zero numbers using while-do-loop in C.

1.3 do-while loop

```
// do-while
do {
   body ;
}
while ( condition ) ;
```

```
int sum = 0, number = 1;
do {
   sum += number;
   ++number;
} while (number <= 1000);
```

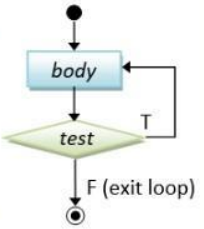

Figure 5: Syntax, example and flowchart of the do-while looping construct.

```c
#include <stdio.h>

int main(void) {

   int n;
   int sum = 0;

   printf("Enter the value of n: ");
   scanf("%d", &n);

   int i = 0;
   do
     {
       sum += i++;
     }while(i<=n);

   printf("The sum of %d numbers is %d\n", n, sum);

     return 0;
}
```

Listing 3: Code to find the sum of first n non-zero numbers using while-do-loop in C.

1.4 Nested Loops

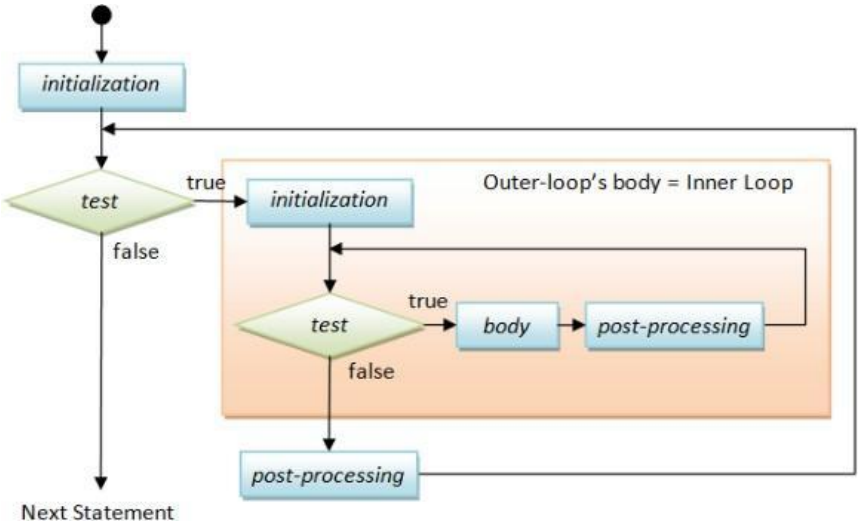

Figure 6: A program which prints a 8-by-8 checker box pattern using nested loops in C.

```
/*
 *  Print square pattern (PrintSquarePattern.c).
 */
#include <stdio.h>

int main() {
   int size = 8, row, col;
   for (row = 1; row <= size; ++row) {      // Outer loop to print all the rows
      for (col = 1; col <= size; ++col) {   // Inner loop to print all the columns of each row
         printf("# ");
      }
      printf("\n");    // A row ended, bring the cursor to the next line
   }

   return 0;
}
```

Listing 4: Code to print a 8-by-8 checker box pattern using nested loops in C.

2 Programming Nuances of the Loop Constructs

2.1 Comparative Study of Looping Constructs

No.	Topics	For loop	While loop	Do...while loop
01	Initialization of condition variable	Before or within the parenthesis of the loop.	Before the loop.	Before the loop or in the body of the loop.
02	Test condition	Before the body of the loop.	Before the body of the loop.	After the body of the loop.
03	Updating the condition variable	After the first execution.	After the first execution.	After the first execution.
04	Type	Entry controlled loop.	Entry controlled loop.	Exit controlled loop.
05	Loop variable	Counter.	Counter.	Sentinel & counter

Figure 7: Comparative study of the different looping constructs in C.

2.2 while-do Vs. do-while

The difference between while-do and do-while lies in the order of the body and condition. In while-do, the condition is tested first. The body will be executed if the condition is true and the process repeats. In do-while, the body is executed and then the condition is tested. Take note that the body of do-while will be executed at least once (vs. possibly zero for while-do).

```
// Input with validity check
bool valid = false;
int number;
do {
  // prompt user to enter an int between 1 and 10
  ......
  // if the number entered is valid, set done to exit the loop
  if (number >=1 && number <= 10) {
     valid = true;
  }
} while (!valid);   // Need a semi-colon to terminate do-while
```

Listing 5: do-while execution of a program that prompts the user for a number between 1 to 10, and checks for valid input.

```
// Game loop
bool gameOver = false;
while (!gameOver) {
   // play the game
   ......
   // Update the game state
   // Set gameOver to true if appropriate to exit the game loop
   ......
}
```

Listing 6: while-do execution of the above program.

2.3 Counter-controlled Loop

```c
/*
 * Sum from 1 to a given upperbound and compute their average (
    SumNumbers.c)
 */
#include <stdio.h>

int main() {
   int sum = 0;      // Store the accumulated sum
   int upperbound;

   printf("Enter the upperbound: ");
   scanf("%d", &upperbound);

   // Sum from 1 to the upperbound
   int number;
   for (number = 1; number <= upperbound; ++number) {
      sum += number;
   }
   printf("Sum is %d\n", sum);
   printf("Average is %.2lf\n", (double)sum / upperbound);

   // Sum only the odd numbers
   int count = 0;      // counts of odd numbers
   sum = 0;            // reset sum
   for (number = 1; number <= upperbound; number = number + 2) {
      ++count;
      sum += number;
   }
   printf("Sum of odd numbers is %d\n", sum);
   printf("Average is %.2lf\n", (double)sum / count);
}
```

Listing 7: Prompt user for an upperbound. Sum the integers from 1 to a given upperbound and compute its average.

2.4 Sentinel-controlled Loop

```c
/* Prompt user for positive integers and display the count, maximum
   ,
   minimum and average. Terminate the input with -1 (StatNumbers.c)
     */
#include <stdio.h>
#include <limits.h>  // for INT_MAX

int main() {
   int numberIn = 0;  // input number (positive integer)
   int count = 0;     // count of inputs, init to 0
   int sum = 0;       // sum of inputs, init to 0
   int max = 0;       // max of inputs, init to minimum
   int min = INT_MAX; // min of inputs, init to maximum (need <
    climits>)
   int sentinel = -1; // Input terminating value

   // Read Inputs until sentinel encountered
   printf("Enter a positive integer or %d to exit: ", sentinel);
   scanf("%d", &numberIn);
   while (numberIn != sentinel) {
      // Check input for positive integer
```

```
19          if (numberIn > 0) {
20             ++count;
21             sum += numberIn;
22             if (max < numberIn) max = numberIn;
23             if (min > numberIn) min = numberIn;
24          } else {
25             printf("error: input must be positive! try again...\n");
26          }
27          printf("Enter a positive integer or %d to exit: ", sentinel);
28          scanf("%d", &numberIn);
29       }
30
31       // Print result
32       printf("\n");
33       printf("Count is %d\n", count);
34       if (count > 0) {
35          printf("Maximum is %d\n", max);
36          printf("Minimum is %d\n", min);
37          printf("Average is %.2lf\n", (double)sum / count);
38       }
39    }
```

Listing 8: Prompt user for positive integers, and display the count, maximum, minimum and average. Terminate when user enters -1.

- In computing, a sentinel value is a special value that indicates the end of data (e.g., a negative value to end a sequence of positive value, end-of-file, null character in the null-terminated string). In this example, we use -1 as the sentinel value to indicate the end of inputs, which is a sequence of positive integers. Instead of hardcoding the value of -1, we use a variable called sentinel for flexibility and ease-of-maintenance.
- Take note of the while-loop pattern in reading the inputs. In this pattern, you need to repeat the prompting and input statement.

3 Interrupting Loop Flow

The break statement breaks out and exits the current (innermost) loop. The continue statement aborts the current iteration and continue to the next itera- tion of the current (innermost) loop. **break** and **continue** are poor structures as they are hard to read and hard to follow. Use them only if absolutely necessary. You can always write the same program without using break and continue.

3.1 break

The break statement breaks out and exits the current (innermost) loop.

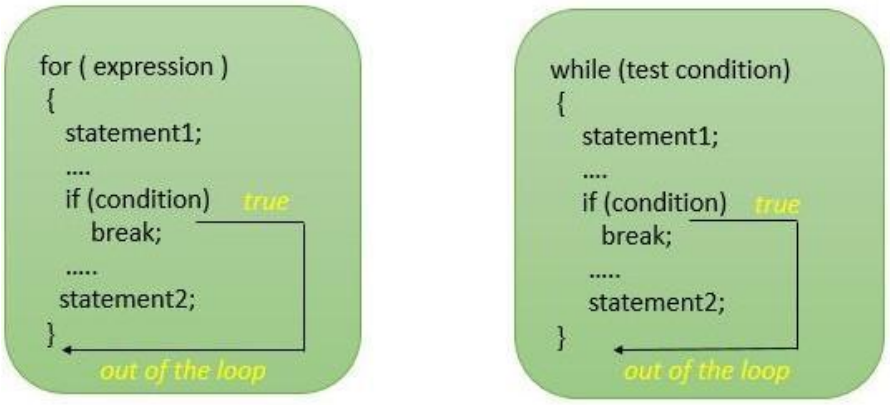

Figure 8: Flow of control in looping structures with break in C.

```c
/*
 *  List non-prime from 1 to an upperbound (NonPrimeList.c).
 */
#include <stdio.h>
#include <math.h>

int main() {
   int upperbound, number, maxFactor, factor;
   printf("Enter the upperbound: ");
   scanf("%d", &upperbound);
   for (number = 2; number <= upperbound; ++number) {
      // Not a prime, if there is a factor between 2 and sqrt(number)
      maxFactor = (int)sqrt(number);
      for (factor = 2; factor <= maxFactor; ++factor) {
         if (number % factor == 0) {   // Factor?
            printf("%d ", number);
            break;   // A factor found, no need to search for more factors
         }
      }
   }
   printf("\n");
   return 0;
}
```

Listing 9: The above program lists the non-prime numbers between 2 and an upperbound.

```c
/*
 *  List primes from 1 to an upperbound (PrimeList.c).
 */
#include <stdio.h>
#include <math.h>

int main() {
   int upperbound, number, maxFactor, isPrime, factor;
   printf("Enter the upperbound: ");
   scanf("%d", &upperbound);

   for (number = 2; number <= upperbound; ++number) {
```

```
13        // Not prime, if there is a factor between 2 and sqrt of
      number
14        maxFactor = (int)sqrt(number);
15        isPrime = 1;
16        factor = 2;
17        while (isPrime && factor <= maxFactor) {
18           if (number % factor == 0) {    // Factor of number?
19              isPrime = 0;
20           }
21           ++factor;
22        }
23        if (isPrime) printf("%d ", number);
24     }
25     printf("\n");
26     return 0;
27  }
```

Listing 10: Rewriting the above program without using break statement. A while loop is used (which is controlled by the boolean flag) instead of for loop with break.

3.2 continue

The continue statement aborts the current iteration and continue to the next iteration of the current (innermost) loop.

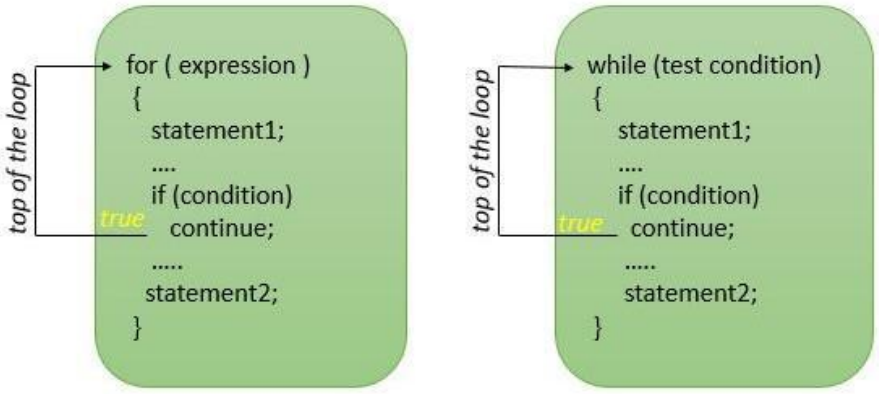

Figure 9: Flow of control in looping structures with continue in C.

```
1  // Sum 1 to upperbound, exclude 11, 22, 33,...
2  int upperbound = 100;
3  int sum = 0;
4  int number;
5  for (number = 1; number <= upperbound; ++number) {
6     if (number % 11 == 0) continue;  // Skip the rest of the loop
       body, continue to the next iteration
7     sum += number;
8  }
9  // It is better to re-write the loop as:
10 for (number = 1; number <= upperbound; ++number) {
11    if (number % 11 != 0) sum += number;
12 }
```

Listing 11: The above program demonstrates the use of continue.

3.3 goto

The goto statement is a jump statement which is sometimes also referred to as unconditional jump statement. The goto statement can be used to jump from anywhere to anywhere within a function. In **forward** reference, the first line tells the compiler to go to or jump to the statement marked as a label. Here label is a user-defined identifier which indicates the target statement. The statement immediately followed after 'label:' is the destination statement. The 'label:' can also appear before the 'goto label;' statement, called the **backward** reference.

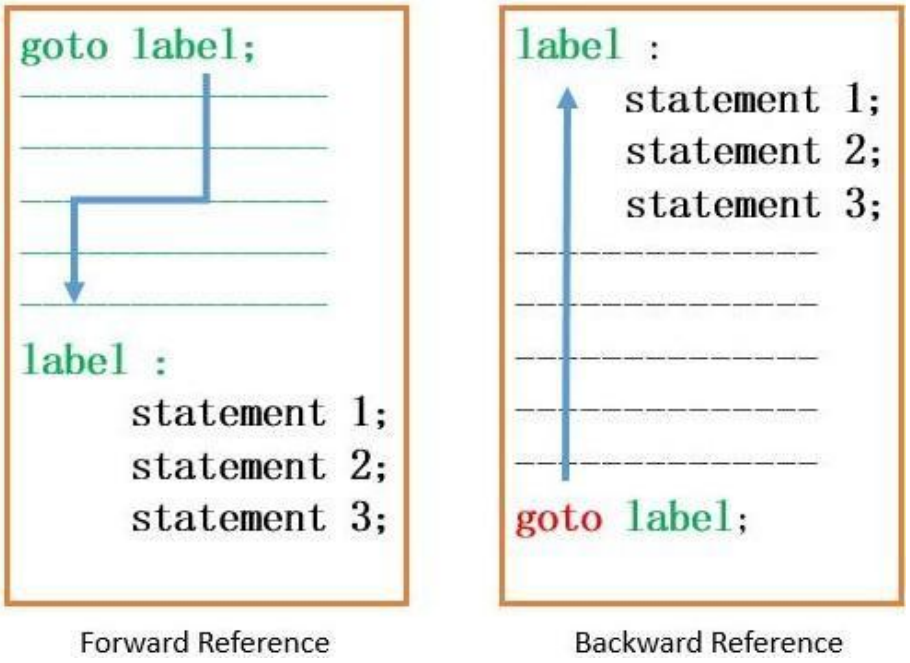

Figure 10: Flow of control in goto-label construct in C.

```c
// C program to check if a number is
// even or not using goto statement
#include <stdio.h>

// function to check even or not
void checkEvenOrNot(int num)
{
  if (num % 2 == 0)
    // jump to even
    goto even;
  else
    // jump to odd
    goto odd;
```

```
14
15 even:
16   printf("%d is even", num);
17   // return if even
18   return;
19 odd:
20   printf("%d is odd", num);
21 }
22
23 int main() {
24   int num = 26;
25   checkEvenOrNot(num);
26   return 0;
27 }
```

Listing 12: Forward reference example for goto.

```
1  // C program to print numbers
2  // from 1 to 10 using goto statement
3  #include <stdio.h>
4
5  // function to print numbers from 1 to 10
6  void printNumbers()
7  {
8      int n = 1;
9  label:
10     printf("%d ",n);
11     n++;
12     if (n <= 10)
13         goto label;
14 }
15
16 // Driver program to test above function
17 int main() {
18     printNumbers();
19     return 0;
20 }
```

Listing 13: Backward reference example for goto.

- The use of goto statement is highly discouraged as it makes the program logic very complex
- The use of goto makes the task of analyzing and verifying the correctness of programs (particularly those involving loops) very difficult
- The use of goto can be simply avoided using break and continue statements

4 Interrupting Program Execution Flow

There are a few ways that you can terminate your program, before reaching the end of the programming statements.

- **exit():** You could invoke the function exit(int exitCode), in stdlib.h, to terminate the program and return the control to the Operating System. By convention, return code of zero indicates normal termination; while a non-zero exitCode (-1) indicates abnormal termination.

- **abort():** The header stdlib.h also provide a function called abort(), which can be used to terminate the program abnormally.

```c
if (errorCount > 10) {
    printf("too many errors\n");
    exit(-1);   // Terminate the program
                // OR abort();
}
```

Listing 14: Terminating a program using exit() or abort().

```c
int main() {
    ...
    if (errorCount > 10) {
        printf("too many errors\n");
        return -1;  // Terminate and return control to OS from main()
    }
    ...
}
```

Listing 15: Using return for transferring control to the OS.

5 TUTORIAL: Trapezium Pattern Interview Question

```
Input  : 4
Output :
1*2*3*4*17*18*19*20
  5*6*7*14*15*16
    8*9*12*13
      10*11


Input : 2
Output :
1*2*5*6
  3*4
```

Figure 11: Sample I/O for the problem.

6 PROBLEMS

6.1 Problem 1

Write a C program to print Pascal triangle upto n rows.

6.2 Problem 2

Write a C program to convert a decimal number input to a number in any given base.

Hint: This could help with figuring out the logic: https://www.codespeedy.com/inter-convert-decimal-and-any-base-using-python/

6.3 Problem 3

Write a C program to find two' s complement of a binary number.

6.4 Problem 4

Write a C program using loops to raise a given number to an exponent i.e. compute $base^{exp}$, taking *base* and *exp* as inputs.

Comment: Does you program work for 0 and negative exponents?

6.5 Problem 5

Write a C program to display the left-arrow-star pattern shown in Figure 12.

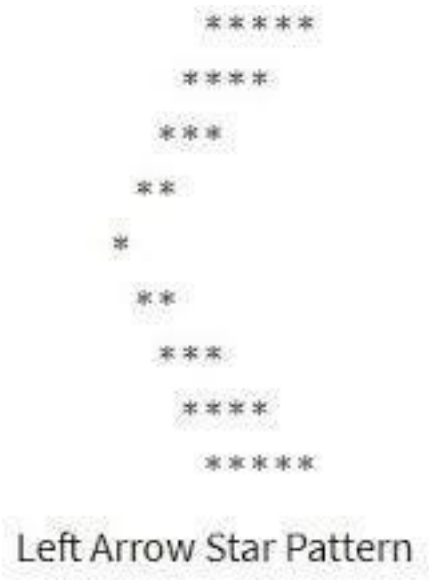

Figure 12: Left Arrow Star Pattern

6.6 Problem 6

Write a C program to display the following number pattern, taking n as input.

For instance, at $n=5$,

555555555
544444445
543333345
543222345
543212345
543222345
543333345
544444445
555555555