

---

## Looping

- Loops enable programmers to tell the computer to repeat a particular block of code multiple times.
  - It is generally impractical to use conditionals a large number of times.
- Consider how a dishwasher might describe their time at work.
  - Unlikely the dishwasher would say “I wash a dish, and then another dish, and then another dish, ...”
  - More like they would say “I washed dishes the entire time I was at work”

## While Loops

- A while loop is the most basic type of loop.
- **while** loops run until a specific controlling condition is not satisfied (i.e. false).
  - The controlling condition is checked *before* the loop executes and every time the loop loops.
- Syntax:

```
1 while(condition){
2   //loop body
3 }
```

- Basic example:

```
1 int a = 1;
2 while (a < 100) {
3   printf("a is %d \n", a);
4   a = a * 2;
5 }
```

- How many times will this loop execute?
  - 7, last time this executes a is set to 128 at the end of the loop
- **A critical note:** something must change in the loop such that the condition is eventually false and the loop exits
  - Otherwise, this is called an infinite loop.
- Consider the following:

```
1 int a = 1;
2 while (42) {
3   a = a * 2;
4 }
```

- 
- The controlling condition in the **while** never changes, and therefore will run forever (since 42 evaluates to true).
  - **break** and **continue**
    - Allows you to control the flow of the loop from within the loop
    - **break** will immediately exit the loop
    - **continue** will skip the remainder of the block and start at the controlling conditional statement again.

```
1 int a = 1;
2 while (42) { // loops until the break statement in the loop is executed
3     printf("a is %d ", a);
4     a = a * 2;
5     if (a > 100) {
6         break;
7     } else if (a == 64) {
8         continue; // Immediately restarts at while, skips next step
9     }
10    printf("a is not 64\n");
11 }
```

- Similar to **if**, you may omit the braces for the block of code associated with the while loop
  - However, this is not recommended for the same reasons as with an **if** statement
  - Grouping of statements is potentially ambiguous (to the programmer, not the computer) that can lead to bugs

```
1 int a = 1;
2 while (a < 100)
3     a = a * 2;
```

- This will just increase **a** until it is above 100
- When a loop ends, the program goes back to the while statement's controlling condition.
  - If the condition is true, the loop executes again
  - If the condition is false, the loop exits
  - The computer does *not* continuously check the controlling condition after each statement in the loop executes. It only checks at the end of every loop
  - If you need to end the loop during the middle of the loop's block, use a **break** to check for the necessary conditions

## For Loops

- Functionally equivalent to a while loop, but people find them to be more readable/maintainable.

- 
- Typically in a while, you'd put some code to modify the controlling condition as the last statement to the while loop (increment, decrement, etc)
    - A for loop moves this to the definition of the loop
  - Syntax:

```
1 for (initialization; controlling condition; loop-ending statement) {  
2     /* code */  
3 }
```

- The *initialization* statement is executed once - at the beginning of the loop
  - Typically, you would assign some variable to be a particular value in this loop section
- The *controlling condition* is the test executed to determine whether or not the loop should run again.
  - It is checked when the loop starts.
- The *loop-ending statement* is typically a form of incrementing/decrementing a value.
  - This statement is executed at the end of every loop statement, but before the controlling condition is checked
  - If you used a **continue** statement, this statement is also executed (i.e. it is not skipped because of the use of a **continue**).
- Any of these may be omitted.
  - You do not have to run an initialization statement
  - You do not have to provide a controlling condition
    - \* What must you do to make sure your loop terminates if this is omitted?
  - You do not have to provide a loop ending statement
    - \* What must you do to make sure your loop terminates if this is omitted?
- Counting example:

```
1 int i;  
2 for (i = 1; i <= 10; i++) {  
3     printf("%d ", i);  
4 }
```

- A for loop can be given no conditions:

```
1 for (;;) {  
2     /* block of statements */  
3 }
```

- This is an infinite loop because it will loop forever unless there is a break statement in the block for the loop

- 
- You may also use the comma operator to add multiple statements inside the loop:

```
1 int i, j, n = 10;
2 for (i = 0, j = 0; i <= n; i++, j += 2) {
3     printf("i = %d , j = %d \n", i, j);
4 }
```

## Do-While Loops

- The do-while loop is the same as a while loop, except the loop controlling condition is checked at the end of the loop rather than at the beginning
- Means the loop is guaranteed to execute at least one time.
- Syntax:

```
1 do {
2     /* do stuff */
3 } while (condition);
```

- Note: the terminating ; is required.
- **break** and **continue** operate the same as with other loops (the controlling condition will still be checked before executing the loop body again when using **continue**)

## Exercises

1. Write a C program to find the sum of first 10 natural numbers.

```
1 #include <stdio.h>
2 void main()
3 {
4     int j, sum = 0;
5
6     printf("The first 10 natural number is :\n");
7
8     for (j = 1; j <= 10; j++)
9     {
10        sum = sum + j;
11        printf("%d ",j);
12    }
13    printf("\nThe Sum is : %d\n", sum);
14 }
```

---

2. Write a program in C to read 10 numbers from keyboard and find their sum and average.

```
1 #include <stdio.h>
2 void main()
3 {
4     int i,n,sum=0;
5     float avg;
6     printf("Input the 10 numbers : \n");
7     for (i=1;i<=10;i++)
8     {
9         printf("Number-%d :",i);
10
11         scanf("%d",&n);
12         sum +=n;
13     }
14     avg=sum/10.0;
15     printf("The sum of 10 no is : %d\nThe Average is : %f\n",sum,avg);
16
17 }
```

- How can we generalize this to allow the user to input a variable amount of numbers?

3. Write a program in C to display the pattern like right angle triangle using an asterisk.

```
1 #include <stdio.h>
2 void main()
3 {
4     int i,j,rows;
5     printf("Input number of rows : ");
6     scanf("%d",&rows);
7     for(i=1;i<=rows;i++)
8     {
9         for(j=1;j<=i;j++)
10        {
11            printf("*");
12        }
13        printf("\n");
14    }
15 }
```

4. Write a C program to determine if a inputted integer is a palindrome

```
1 #include <stdio.h>
2
```

---

```
3 int main()
4 {
5     int n, num, digit, rev = 0;
6
7     printf("Enter a positive number: ");
8     scanf("%d", &num);
9
10    n = num;
11
12    do
13    {
14        digit = num % 10;
15        rev = (rev * 10) + digit;
16        num = num / 10;
17    } while (num != 0);
18
19    printf("The reverse of the number is: %d\n", rev);
20
21    if (n == rev)
22        printf("The number is a palindrome\n");
23    else
24        printf("The number is not a palindrome\n");
25
26    return 0;
27 }
```