Looping

- Loops enable programmers to tell the computer to repeat a particular block of code multiple times.
  - Is 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 watch 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:

```
while(condition){
  //loop body
}
```

- Basic example:

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

- 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:

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

- 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.

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

- 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

```c
int a = 1;
while (a < 100)
  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:

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

- 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:

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

- A for loop can be given no conditions:

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

- 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:

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

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:

```c
do {
  /* do stuff */
} while (condition);
```

- Note: he 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.

```c
#include <stdio.h>
void main()
{
  int  j, sum = 0;

  printf("The first 10 natural number is :\n");

  for (j = 1; j <= 10; j++)
  {
    sum = sum + j;
    printf("%d ",j);
  }
  printf("\nThe Sum is : %d\n", sum);
}
```

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

```c
#include <stdio.h>
void main()
{
  int i,n,sum=0;
    float avg;
    printf("Input the 10 numbers : \n");
    for (i=1;i<=10;i++)
    {
    printf("Number-%d :",i);

        scanf("%d",&n);
        sum +=n;
    }
    avg=sum/10.0;
    printf("The sum of 10 no is : %d\nThe Average is : %f\n",sum,avg);

}
```

- 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.

```c
#include <stdio.h>
void main()
{
  int i,j,rows;
  printf("Input number of rows : ");
  scanf("%d",&rows);
  for(i=1;i<=rows;i++)
  {
      for(j=1;j<=i;j++)
      {
        printf("*");
      }
      printf("\n");
  }
}
```

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

```c
#include <stdio.h>

```

```c
int main()
{
  int n, num, digit, rev = 0;

  printf("Enter a positive number: ");
  scanf("%d", &num);

  n = num;

  do
  {
    digit = num % 10;
    rev = (rev * 10) + digit;
    num = num / 10;
  } while (num != 0);

  printf("The reverse of the number is: %d\n", rev);

  if (n == rev)
    printf("The number is a palindrome\n");
  else
    printf("The number is not a palindrome\n");

  return 0;
}
```