Guided Introduction to Programming
Lesson 4: Loops
Teacher's Guide

Welcome back to the Teachers Edition of a Guided Intro to Programming. This is the final lesson on the basics of programming in C and after this you should be able to do a variety of simple problems that utilize math, printing and scanning, and if and else statements. Today's lesson will be on loops which are simple functions but highly powerful as they cut down on programming time if you need to check or add together vast quantities of items.

First, students should define what a loop is. In simple terms, a loop is a function that, similar to and if and else statement, takes a condition and repeats a function or set of functions in the loop until that condition is met. It is like running laps around a track, if you need to run 4 laps, you begin to run, and you complete your first lap. You have not reached 4 laps yet, so you do it again, and again, and again until you reach 4 laps. What you just did was a loop, you completed a function, but the condition was not met so you ran it again until the condition was met.
Once you understand what a loop is and how it works, next compare the types of loops. There are many types of loops but today the focus will be on two types, the for loop and the while loop. Though a for loop and a while loop have the same function their use is determined by whatever task you are programming or to general preference. Some programmers make their choice based on which one they feel more comfortable using. In general, a while loop is used if you want an action to repeat itself until a certain condition is met i.e., if statement. A for loop is used when you want to iterate through an object. i.e., iterate through an array. You should use a for loop when you know how many times the loop should run. If you want the loop to break based on a condition other than the number of times it runs, you should use a while loop.

As a note to what kinds of loops there are, we have: do while loops, while loops, for loops, and recursive loops which occur when you call a function recursively. This is just in case a student asks what other loops there are.

A while loop does a task while a condition is true. For example: While (X<10){ X++; }.
What I just wrote in terms we will understand is while X is less than 10 add 1 to X every iteration. An iteration is repetition of a computer procedure, so in simple terms it is just the program repeating itself until completion. You may be asking right now, "What can I do with a while loop?", and that is a good question. Like any loop you can use it to repeat a task until a condition is met. A simple example of this is like when you are playing a video game and you are holding down a button to do an action. The programmers of the game can make one press of the button do one action or they can use a while loop to detect when you let go of the button to cancel the action. However, there is much more advanced programming involved in that so we

will save that for another time. For now, let us look at a very famous type of while loop, the prime number solver.

To show off the *for loop*, I want to use a simple prime number solver, if your students have not yet learned about prime numbers, I have included an explanation of what prime numbers are below. Please read over it and if you deem it necessary, please feel free to explain in your own way what prime numbers are. I know everyone learns math differently, but this was the best technical definition I could come up with.

First, examine while loops. A while loop does a task while a condition is true.
For example: While (X<10){ X++; }
This translates as:
While X is less than 10 add 1 to X every iteration.
An iteration is repetition of a computer procedure, so in simple terms it is just the program repeating itself until completion.

So then, what can be done with a *while loop*? Like any loop you can use it to repeat a task until a condition is met. A simple example of this is like when you are playing a video game and you are holding down a button to do an action. The programmers of the game can make one press of the button do one action or they can use a while loop to detect when you let go of the button to cancel the action. However, there is much more advanced programming involved in that function that will not be covered in this lesson. For now, we will examine at a very famous type of while loop, the prime number solver.

You may have learned about prime numbers in class but even if you have not you can continue with this lesson. In simple terms, prime numbers are numbers that are not a product of two smaller numbers.
For example, 17 is a prime number because the only numbers that can multiply to get 17 is 17 and 1.
In contrast 18 is not a prime number as you can multiply 3 and 6 to get 18.
How can we tell whether a number is prime using a *while loop*? Without a loop it is very time consuming to determine if a number is prime because without a loop you would have to check each number manually. This method is not efficient. Code can be used to create a program to detect prime numbers.

```
#include <stdio.h>

int main(void) {

  int x = 0;
  //We start count at 2 because every number is divisible by 1
  int count = 2;
```

```
  printf("Please enter a number: ");

  scanf("%d", &x);

  //Puts is just used to create a new line easier
  puts("");

  printf("This while loop will detect if %d is prime.\n", x);

  while(count < x){
    if(x % count == 0){
      printf("%d is divisible by %d, %d is not prime.\n", x, count, x);
      //Return 0 is used here to stop the program as it has detected that x is
not prime.
      return 0;
    }else{
      printf("%d is not divisible by %d.\n", x, count);
      count++;
    }
  }

  printf("Based on the while loop, %d is a prime number.\n", x);

  return 0;
}
```

When you input a number, this code will tell you the numbers it is divisible by. The code will also tell you if that number is prime. For the sake of consistency, in the example below the same numbers are used as a test from earlier. Image 1 shows the output of the program when you input 18:

```
clang-7 -pthread -lm -o main main.c
./main
Please enter a number: 18

This while loop will detect if 18 is prime.
18 is divisible by 2, 18 is not prime.
```

As you can see, the code was able to detect if 18 was prime early in the program because 18 is divisible by 2.

```
clang-7 -pthread -lm -o main main.c
./main
Please enter a number: 17

This while loop will detect if 17 is prime.
17 is not divisible by 2.
17 is not divisible by 3.
17 is not divisible by 4.
17 is not divisible by 5.
17 is not divisible by 6.
17 is not divisible by 7.
17 is not divisible by 8.
17 is not divisible by 9.
17 is not divisible by 10.
17 is not divisible by 11.
17 is not divisible by 12.
17 is not divisible by 13.
17 is not divisible by 14.
17 is not divisible by 15.
17 is not divisible by 16.
Based on the while loop, 17 is a prime number.
```

There is a reason why the code only runs the program up to the input number (17). This is because, like the number 1, we know that whatever number we put in is divisible by itself so we do not need to check that, otherwise it would break the logic in the loop and would have to rewrite some parts of the program.

An important thing to look for when your students are making loops is the condition they provide. Say for instance I wanted to print every number up to 17 starting with 1. The proper way to write it would be X <= 17, which will start at X = 1 and go until it gets to 17. However, if they write it as X < 17 then it would only go up to 16. A small but important detail when trying to get proper inputs.

Most of the time the different types of loops can do the same tasks even with a little variation in the code, but if done correctly will have the same output. After you learn about the for loop and how it is written, try writing a prime detector program using the for loop, use the code provided above as a guide if you are still a little fuzzy on the math.
Here is the code for the Prime Number *For Loop*:

```c
#include <stdio.h>

int main(void) {
```

```
int x = 0;

  printf("Please enter a number to see if it is prime: ");
  scanf("%d", &x);
  puts("");

  for(int i = 2; i < x; i++){
    if(x % i == 0){
      printf("%d is not prime as it is divisible by %d\n", x, i);
      return 0;
    }
  }

  printf("%d is prime.\n", x);

  return 0;
}
```

First, a *for loop* has the same function as a *while loop*, as they are both loops. However, a *for loop* typically operates with counters and mostly numbers as conditions. This is different from true or false logic statements that were introduced in the logic lesson.

The best way to explain is to go through a quick example:
A *for loop* has three things it needs when making the loop and those are: an initialization, a condition, and an updater.

1. An initialization is the creation of a counter such as: int count = 0; we use this to iterate or loop through a set number of times based on the condition.
2. The condition is the same as a while loop, an example of a for loop condition could be: count < 20.
3. Without an updater the loop would either error out or just go on indefinitely, therefore the count needs to be updated with every pass through the loop with something like: count++ which just adds 1 to count.

Given these three circumstances, a *for loop* will look something like this:

for (initialization; condition; updater) {
Body of Loop
}

Keep in mind about how a condition is written. If we changed what is below to i <= 10 without changing the initial value of i, then the loop would operate 11 times rather than 10 which is not our desired output.

As for actual code, just print hello 10 times:

```c
#include <stdio.h>

int main(void) {

  for(int i = 0; i < 10; i++){
    printf("Hello\n");
  }

  return 0;
}
```

```
clang-7 -pthread -lm -o main main.c
./main
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
```

You can see the program has printed out “Hello” 10 times. This is far easier than writing the print function 10 times manually. An important thing to note for both loops is that they both need an updater. The *for loop* has its updater in the creation of the function like what is shown above in Image 3. However, the *while loop* does not have its updater in the creation of the function, therefore we need an updater in the body of the function. If there is no updater in either function the loop will just go on indefinitely which is why the code needs to have an appropriate updater.

A more practical application of loops is the calculation of a factorial. I am certain that you have not seen a factorial yet, but if you have that is great. The factorial is an important mathematical concept that comes up in the more rigorous high school math classes and in college, so it is good to get a foundation for how it works early. A factorial is the product of all positive integers less than or equal to a given positive integer and denoted by that integer and an exclamation point. For example, 5 factorials, looks like 5! is 5*4*3*2*1 = 120. Not too complicated, we can make this process easy on a computer with a simple for loop.

Similar to explaining prime numbers, please look over the explanation of a factorial as your students may need a different explanation to fully understand what a factorial is.
The code for a factorial, is down below:

```c
#include <stdio.h>

int main(void) {

int x = 0;
int sum = 1;

printf("Enter a number and this program will return its factorial.\n");
scanf("%d", &x);

  for(int i = 1; i <= x; i++){
   printf("%d * %d = ", sum, i);

   sum = sum * i;

   printf("%d\n", sum);
  }

printf("%d factorial is %d", x, sum);

  return 0;
}
```

```
clang-7 -pthread -lm -o main main.c
./main
Enter a number and this program will return its factorial.
5
1 * 1 = 1
1 * 2 = 2
2 * 3 = 6
6 * 4 = 24
24 * 5 = 120
5 factorial is 120
```

As you can see that code works, feel free to copy that code and try it out. Keep in mind you can only do factorials up to a certain number because after that the number becomes so big that the computer cannot compute it properly. This is due to memory sizes and other more complicated architecture properties of modern computers.

As a bit of practice try and do a factorial computation with a *while loop*, as it is possible to do so. Feel free to use the code above as a reference.

The code for the *Factorial While loop* is:

```
#include <stdio.h>

int main(void) {

  int x = 0;
  int sum = 1;

  printf("Please enter a number to see its factorial: ");
  scanf("%d", &x);
  puts("");

  while(x > 0){
    printf("%d * %d = ", sum, x);

    sum = sum * x;

    printf("%d\n", sum);

    x--;
  }

  return 0;
}
```

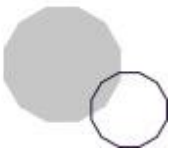
With that said and done you have completed this lesson,

Congratulations

Glossary

Counter – a variable used in loops to keep track of how many iterations we have gone through, used to close loops once we reach a certain number of iterations.

Efficient – efficiency is a peak level of performance that uses the least number of inputs to achieve the highest amount of output.



Factorial- the product of an integer and all the integers below it

Iteration - the repetition of a computer procedure.

Initialization - set (variables, counters, switches, etc.) to their starting values at the beginning of a program.

Loop - a function that, like and if and else statement, takes a condition and repeats a function or set of functions in the loop until that condition is met.

Prime Number - numbers that are not a product of two smaller numbers, i.e., 5, 7, 13, etc.…

Updater – code that updates the counter with every pass through the loop.

Assessment

1. Use a for loop to print out all the numbers from 1 to 1000.
2. Use a while loop to print out all the numbers from 1 to 1000.
3. If a for loop that looks like this:

   For(int I = 0; I <20; I++)

   {Printf("%d\n", I);}

   How many times will the loop run through?
4. Use a for loop instead of a while loop with the code used to find prime numbers in Part 2.

Extensions and/or Additional Resources

Remember to keep practicing your code and if you are interested use these links to keep learning past what we have done or just feel free to look up any programming sites that can teach you as there are thousands of great sites that have comprehensive lessons and descriptions on how to code.

- Link to Lesson 4 Video

- Link to Programming Vocabulary
- https://www.w3schools.com/
- https://www.geeksforgeeks.org/c-programming-language/
- https://replit.com/languages/c

Thank you for taking the time to learn code with me as your guide and I hope to see you coding in the future.

Created By:
Ryan Floyd, Computer Science, '22

Thank you for downloading this lesson, please take a moment to complete our survey