Week 3 Module 6 Loops/Iterative Statements

Motivation 1.2

Hi there hope you are having a great day. Let's recap what we've seen so far. So we've talked about three fundamental constructs of C++. We've talked about data expressions control flow. Regarding data we've covered a few built in data types such as int unsigned int float double char bool all of that. Regarding expressions we've talked about arithmetic expressions we've talked about Boolean expressions. And when talking about control flow so we've seen the default sequential flow where statements are executed one after the other and we've also talked about branching such as IFs IF ELSE and so on. Let's try to use all of that in order to solve actually it seems a very simple problem. We would read positive int n from the user and print the numbers from one up to n. So for instance if the user enters I don't know 4 then the program would respond by printing 1 2 3 4. Ok let's try thinking how we can do all of that. So it seems like sequential flow can't really work so we can't do cout 1 cout 2 cout 3 cout because we don't know where to stop. And actually branching also seems kind of impractical. I don't know maybe we can do something like if n equals 1 cout 1. Else if n equals 2 cout 1 and 2. Else if n equals 3 cout 1 2 and 3 and so on but since our program needs to have a finite amount of lines of code we can't make it work for all inputs. So what we have so far seems not enough in order to implement this kind of simple requirement. And to do that we would need a new kind of control flow that would help us iterate and repeat statements over and over. And that's basically what we are going to be talking about today. I am going to show you two statements for repetitive executions for iterative executions. The first one is going to be a while.

While Loops 2.1

Ok so the syntax of the while statement goes like this. So you have a few statements then comes the while keyword and then enclosed in parentheses you have a condition. And as a compound statement you can have a body of statements that are tabbed one tab to the right. After the while you have your program just continues at the original alignment. But that's basically the syntax of a while expression. Let's talk about the semantics how a while expression is basically a while statement is basically executed. So the program is executed up to the while statement. When it reaches the while statement the first thing that happens is that condition is evaluated and then actually there are two cases. Either it is true or false. If the condition is true the body is executed and the condition is reevaluated. Again two options either true or false. If it is true body is executed condition evaluated once more. True body condition. True body condition evaluated. When the condition turns false the program just continues in its original path. So basically while the condition is still true the body is executed over and over and over. That's why it is called the while statement. While the condition is true we keep repeating the execution of the body over and over. When the condition turns false we break out and continue with the flow of our program. So that's basically the semantics of a while statement. This is syntactically the extended version of a while. The simpler version just as we had with the Ifs doesn't require a compound statement. You can have a single statement body then you don't need the curly braces to enclose this statement in it.

Solve Let's Count Program (using While) 2.2

Ok let's then try to use this while statements this iterative capability that we just talked about in order to solve our counting program. Let's try and think how we can use the capability of repeating some statements over and over in order to print numbers let's say from 1 to N. So we can maybe maintain some kind of a variable a counter that would be initialized to one printed and incremented. Printed and incremented. Something like that we will start counter as one and we will print one and increment counter to two. We will print two and we will increment counter to three. Will print three and increment counter to four. So basically we are repeating two statements here print the value of counter and increment it. Print the value of counter and increment it. If we do it a right amount of times we will basically print numbers from 1 to wherever we decide to go. Let's implement it using C code now.

For Loops 3.1

Ok so that was kind of cool being able to repeat stuff over and over. We can do that using a while statement. There is another control flow for iterative statements which is called the for statement. Let me again show you the syntax and the semantics of for statements. So syntactically it goes something like that you have your program and then you have the for key word. Enclosed in parentheses you have three parts that are separated by semi colons. You have the initialization statement semi colon condition statement and semi colon and increment statement. So you have these three parts that are enclosed in parentheses. After that you have compound statement of the body of the for loop which is again spaced one tab to the right that's basically the for statement. After that we have our program that continues at the original alignment. Semantically the way a for loop basically works is something like that. You execute your program up to for statement and then when the execution reaches the for statement first thing it does is executes the initialization statement one time. After executing this statement the condition is evaluated two options either it is true or false. If the condition is true the body is executed followed by the increment statement. Once again the condition would be evaluated. If it is true body increment. Condition is evaluated if it is true body increment. Condition is evaluated if it is false the for basically ends and the program just continues with its original execution. So that's for syntax we have the simplified for where you don't need the compound statement you can have a single statement that would be repeated each iteration. Then you won't need the curly braces. At first it seems like for statement are much more complex than while as we practice to use fors you will see that they have their advantages. They are sometimes more readable and preferred over whiles. But we will have to get used to that.

Solve Let's Count Program (using for) 3.2

So now let's try to solve the same problem here of counting from 1 to N but this time let's do it using a for loop.

Counting up using for.mp3

Ok then let's try to count up now using the for loop. So once again let's first ask the user to please enter a positive integer let's read this input so we declared an n and read into that n and now we're supposed to start counting from one to n. Once again we'll create this counter variable but now that we have a for loop as we said there are three parts of this for statement there is the initialization the condition and the increment. So we can initialize counter to one we want to keep repeating as long as counter is less or equal to four. And then we want to repeat two things we want to print the value of counter and we want to increment counter. Instead of incrementing counter here inside the for loop we can increment it in the increment position so counter plus plus goes over here. Let's try to trace this execution so once again we have our two variables n and counter. Let's assume the user enters four for n. And first we initialize counter to one we check whether the Boolean condition is true or false if counter is less than four in this case it is. So we are printing one the value of counter. And after executing the body we execute the increment statement basically increasing counter to two. We then check whether the Boolean expression is still true it is so we print value of counter and we increment counter to three. The Boolean condition is still true so we print the value of counter and increment counter basically turning it to four. The Boolean expression is still true so we are printing the value of counter printing four and incrementing the value of counter basically incrementing it to five. Now the Boolean condition is false we are breaking out of the for loop we can once again return zero and end this execution.

Let's Count Problem- Comparing the Two Solutions 3.3

If we take a minute to compare these two implementations of the counting up program. One using the while loop the other using the for loop we'll see ok so obviously they both read an n from the user. But we see that they both implemented the same idea of maintaining a counter initialize to one and repeating print statement of the current value of counter and increasing it over and over four times. The difference as you can see is the location or the way this counter is basically managed in each program. So they both initialized counter they both go up to when counter is less or equal to four. They both repeat printing the value of counter and incrementing it. In the while loop the management of this counter is split into three locations one for initializing one for checking the condition and another for the increment. Where in the for loop they are all kind of binded together in the for loop header where we do all the management of counter in a single location in a single place. We initialize we check the condition and we increment all at the same place. That is one of the advantages of a for loop over a while that it is in that way more readable to a programmer that you can easily see how this loop is basically managed.

Counting & Summing Digits 4.1

Let's try to solve another program using iterative statements. Let's try to solve a problem that we first read a positive integer from the user. And then try to calculate the number of digits in this number and what the sum of these digits. For example we can ask the user to enter a positive integer. The user would enter let's say 375 and the program would respond by saying that 375 has three digits. Yea because 375 has three digits and their sum is 15 and that's because 3 plus 7 plus 5 equals 15. So given a positive integer we would have to figure out how many digits are there in this number and what their sum is. Let's try to think how we can do that kind of a thing. So for that I think we should try to iterate over the digits each time we can count a digit and add it to some accumulator that would hold the total sum. So we would need two variables count digits variable and a sum digits variable. We would initialize both of them to zero and then we would start iterating over the digits one by one. We will start let's say with 5 and for each digit we will count it and we will add it. So counting 5 would turn count digits to 1 and adding 5 would turn sum digits into 5. And then we would go over the next digit 7 we will count it and add it would make count digits 2 and sum digits would be 12. Five plus seven. Next digit we will count it makes count digit 3 and we will add it makes sum digits 15. We will repeat this process over the digits each time counting it and adding it. Two things that are basically repeated. That's the basic idea of counting and summing up digits in the number. The only challenge we have left is figuring out how to iterate over the digits. We can't access each digit separately so we can think of mathematical property of integer numbers that can come in handy for this. And that's for example if we take 375 and divide it by 10 we will get 37 and the remainder of 5. We can then take 35 mod it by 10 to get the 5 and 375 div 10 would give us the 37. Basically splitting the ones digit from the rest of the digits. This way we can isolate the ones digit work and accumulate its data and then remove it and keep working with the rest of the digits. This trick here works not only for three digit numbers it works for any length of a number. For example let's take a four digit number 2375 divided by 10 we will get 237 and the remainder of 5. Once again 2375 mod 10 would give us the ones digit the 5. 2375 div 10 would remove this 5 would of the number and we will be left with 237. This thing here as we said works for any length of a number. Taking n plus 1 digit number an to a zero mod 10 would isolate the ones digit a0. And div it then by 10 would isolate the rest of the digits basically removing a zero from the number keeping us left with an to a1. Let's take all of that together in order to better explain or better understand the process of counting and summing up the digits. So we said we have our number 375 both accumulators count digits and sum digits that would be initialized to zero. We would also maintain the current digit for each iteration. And then each iteration we would isolate the digit count it add it and remove it. For example first iteration we would isolate ones digit the 5 we will count it into the count digit we will add it to the sum digits and we will remove it from the 375. Keeping us or having us left with 37. And then again these four statements would be repeated. Isolating the ones digit the current digit would be 7. Counting it adding it and removing it. And the four statements would be repeated again taking out the ones digit counting it summing it and removing it. If we keep on the repeating these four statements over and over basically we are iterating over the digits one by one counting each digit and accumulating its sum. Let's try to implement that using C++.

Counting and Summing Digits 4.2

Ok so let's try to write this program. So let's have our main let's keep some space for variables. First we need to read the input from the user let's ask the user to enter a positive integer so enter a positive integer let's also break the line. And then let's read the user's input so let's have a num integer variable let's cin into this num. And now we should start iterating over the digits of num accumalting their sum and counting each digit. We said we were going to use two accumulating variables one for the amount of digits one for the sum of the digits. So let's declare sum digits and count digits these would be our accumulating variables. Before we start iterating we have to initialize them so let's set sum digits to zero and also count digits to zero and now we should start iterating. We said since we don't know how many iterations we are going to have we'll use a while loop here so while again let's keep the put in condition that controls the while loop for a later stage. Let's first figure out the statement there are going to be repeated each iteration. So we said that each time we want to extract the ones digit add it to the sum digits and count it over and over. So let's have a current digit variable that would hold the current digit we are dealing with in this iteration so here is the current digit. So first thing we want to do is take out the ones digit. We said we were going to use mod 10 for doing exactly that so current digit gets num mod 10. So now that we have the ones digit we should count it and add it to sum. So we should increase count digits let's increment it plus plus and let's add a current digit value into the sum digits variable. Sum digits plus equals our current digit. After doing that we need to remove the ones digit from num we said we'll do that by diving num by 10. So let's set num to the new value after removing the ones digit. So let's set nums digit equals num div 10. So these are basically the four statement we want to repeat each we want to isolate the ones digit we want to count it we want to add it and we want to remove it from them. Over and over this way we are accumulating count the digits fro num. So that basically seems to be the statements inside the body of the while in order to figure out what is the Boolean condition that controls this while loop let's try to trace this execution and during this trace we'll try to figure out how to phrase the Boolean condition. So let's have our memory we have a few variables we have num sum digits count digits and current digit. First we ask the user to enter a positive integer the user enters let's say 375 then we set some digits to zero and count digits to zero and now we start iterating. So let's assume the Boolean condition is true and start repeating these four statement over and over. So first iteration current digits gets num mod 10 375 mod 10 with a remainder when we are dividing 375 by 10 would be five. So our current digit would be five which makes sense that's exactly what we wanted to do. Then we are counting this digit basically incrementing count digits and we are adding five to the value of sum digits basically turning sum digits to be five. After we are doing that num becomes num div 10 so first we evaluate num div 10 which is 37 and num then is assigned to 37 basically removing the ones digit from 375. That ends the first iteration. Second iteration once again current digit is 37 in this case mod 10 that would be 7 right the remainder when we are dividing 37 by 10 is 7. We are counting this digits basically incrementing count digits and we are adding seven to the sum digits that would make sum digits be 12. Last thing is num div 10 so when 37 is divided by 10 we get 3 and we still want to have one more iteration for adding this 3. So current digit would be num mod 10 just so we won't make any mistakes when we divide 3 by 10 we get 3 so the mod would return 3 and that's exactly what we want current digit to be. That's the digit we are going over this iteration we are counting it basically incrementing count digits and we are adding it to sum digits making sum digits 15. Finally we are assigning num to num div 10 and num div 10 in this case is zero so that completes the third and hopefully the last iteration of the while loop. And I think at this point we should be able to say what the Boolean condition for the while in order to control this statement to have only three iterations in case of 375. So it was supposed to be true up to now and it is a Boolean condition should turn to false exactly at this point because we don't want anymore iterations. So we see that when num turns to zero that's where we want to stop we don't want to have more iterations basically when num is positive we want to repeat these three statements. When it was 375 we want to make our first iterations when it was 37 we wanted to make our second iterations and finally when it was 3 we wanted to make our last iteration. When it turned zero we wanted to break out of this while loop so while num is positive we want to repeat when we break out of this loop now we have the number of digits and their sum at our possession so we just need to announce it to the user. So let's cout let's say something like 375 has three digits and their sum is 15. So it would be num has count digits digits and their sum is now let's just strain the value of the sum digits variable sum digits let's break the line here return zero and that basically concludes this execution.

Computing the Average 5.1

We've talked about for loops while loops iterative statements. Let's look at another problem now. Let's talk about how to compute the average of a collection of numbers. For example let's assume the user enters a sequence of grades and calculates and prints the average of the sequence. Let's try to aim for a behavior that looks something like that. At first the program would ask the user please enter the number of students you have in the class. The user can say let's say 4 and then the program would ask the user please enter the students' grades separated by spaces. The user would then enter four grades for example 71 86 68 94. And eventually the program would respond by saying what's the average of this sequence. In this case the class average is 79.75. So in order to implement this problem here we need to calculate an average. And in order to calculate the average basically you need to sum up all the numbers and divide it by the amount of numbers. That would give you the average. So to do that we know how many numbers are going to be that's the first input we got from the user. The number of students in the class but in order to figure out the sum of the numbers the sum of the grades we would need to accumulate them in some kind of sum variable. Just as we did in the calculating the sum of the digits of a number. So basically we are going to repeat here two things over and over. We will read an input and add it to sum. Read an input and add it to sum. Read an input and add it to sum. Over and over for each grade the user enters. After we will finish up reading and accumulating the sum we will have a variable containing the total sum which we can then divide by the amount of numbers to figure out what the average is. Let's get into the details of this implementation.

Computing the Average Screen Share 5.2

First we need to read the number of students. So let's create a variable num of students. And let's ask the user please enter the number of students in class and end the line. And just read it into this num of students. Then we should announce to the user to start entering the sequence of grades. So let's say enter the students' grades. Let's also ask the user to separate them by space. Ok I think that's good enough instructions here for the user. Ok so now we should start reading the numbers and adding them up. For that we would need some kind of iterative process. Since we know exactly how many iterations we are going to have. We are going to have num of students iterations. A for loop here is a good choice. So let's create some counter variable initialize it to 1 and each iteration will just increment this count. And if we want to do num of students iteration starting a 1 incrementing it by 1 each time we just need to make sure that count is less or equal to num of students in order to keep on going. Let's just declare this count variable. So this for loop header here basically controls process that does num of students iterations. And then we said that each iteration we want to read an input. Let's read it into current variable and to add it to some accumulator. So we should let's first declare current. We also need some variable that actually should be initialized before we start iterating. So first it is initialized to zero and then each iteration we just add to sum the value of the current element we read. So if we take a second look at this piece of code here we first initialize sum to zero and then we start iterating num of students times. Each time we read an input and add it to sum over and over. Read an input add it to sum. At the end we would have the total sum of all the grades the user entered. We can then calculate the average as divide sum by a number of students. Let's create a variable for the average. Even though the grades are integers and number of students is integer. The average could be a number with a fractional part so for that I would define the average variable as a double. And then I would just assign average to sum divided by num of students. Then again a closer look here would suggest that the slash operator in the context of dividing two integers won't do a real division but would basically do integer division div. So in order to make it a real division we would need to cast both sum and number of students to double. Now average is the result of really dividing a sum by number of students. We should just now announce that the average is average is and then we will just print the value of the variable average and end the line. That basically should be the program. Let's test it out. Compile it and execute it. No bugs here which is good. Enter the number of students so we have four students. Enter the grades let's just do I don't know 89 75 93 and 78. And then the average is 83.75 I believe that's the average here. Looks good.

Computing the Average 5.3

We've calculated the average which was a cool thing. But taking a closer look here the behavior of this program we can see that we are asking the user to say in advance how many numbers are going to be in the sequence. Which is maybe fine in this case but in a lot of other cases it is a not so user friendly and sometimes it is basically impractical for the user to know in advance how many or how large it is the input. A lot of times we just want the user to keep on entering the input as they find it and to signal in some way that they are done entering the input. Let's revise our program a bit to allow this kind of behavior. Let's ask the user to enter a sequence of grades and just type negative one which is obviously not a valid grade to signal that the input is over. So the behavior should be something like that. First the program would say what the instructions are. Enter the grades separated by space and your sequence by typing negative one. Then the user would start entering the sequence of grades in this case 71 space 86 space 68 and so on 94 and then negative one which is just saying I am done entering the grades. The program would then know to announce the average which is in this case 79.75. Let's take a minute and think how we can implement this kind of a behavior. It seems like we can't use a simple for loop here as we have used before going with a count from one to the amount of numbers in our sequence. Just because we don't know how many numbers there are going to be in the sequence. But we still want to do some kind of repetitive process of reading and accumulating reading and adding reading and adding. In order to calculate the total sum of the numbers. So it seems like a for loop won't be the control flow we are going to choose. In this case we are going to choose a while loop. And another technique I want to show you here is the use of variables which are named flag variables. Flags are basically two state variables they are either down or raised or up. We can use them in this type of situation something like that. We will initialize the flag to be down and then each time we read an input we will add it by the way we also need to count the number of elements in order to or be able to divide the sum by the amount. In this case we would need to accumulate both the sum and a count of how many elements there are. So each iteration we will read a number add it and count it. Read a number add it and count it. The flag would remain down as long as we read grades. We read a grade add it and count it. Read a grade add it and count it. As we see negative one we just raise the flag basically signally to break out of this repetitive process and to calculate the average. At that point in time we would have accumulated the total sum we would have accumulated the number of elements that the user entered. And then we would be able to divide them one by the other. So that would be the big idea here of our implementation. Just one thing before we move on really implementing is how to implement this two state variable. And it seems very reasonable to use Boolean typed variable because a Boolean also has two states false and true. Basically corresponding to the two states of a flag, down and up. So the down would be let's say false and up would be true and then using this Boolean variable we can use it in the while condition to control this repetitive process. Let's go down and implement this here.

Computing the Average Screen Share 2 5.4

Ok so let's implement this program here. So first let's tell the user the instructions so cout enter the grades separated by space. Let's have it in two separated lines. So let's end one line here and now let's just tell the user enter the sequence by typing negative one and break the line. Ok so now we should start reading the numbers and adding them and counting them. So we would have as we said a while loop right. But in order to control this while loop we said we are going to use flag which is a Boolean variable. We won't name it flag. Let's try to find a name that more represents what this flag basically stores. Let's name this flag maybe seen end of input. When you choose a name for a flag or for a Boolean variable I think it is a good practice to choose a name that you want to respond by saying yes or no. Seen end of input is either yes or no true or false. So that's a reasonable name for a flag. We said we have to initialize seen end of input to false. And basically we want to keep on repeating the statements in the while loop while seen end of input is still false. When it turns to true we want to break out. So this is basically the structure of our program here. We just need here to read the input add it count it. Read add count and so on. In order to accumulate the sum and the amount of numbers we would use two variables. One for the sum and one for number of students. We would need to initialize both of them before we start iterating. So sum would be initialized to 0 and number of students would also be initialized to 0. And then we should just each iteration read the current input let's just declare current. And we should add it and count it right. But we should be careful here because actually all the grades should be added and counted but if the input is negative one we basically want to break out of the loop. So before adding to sum and counting by incrementing number of students let's check if our current input is negative one. In this case double equal obviously. In this case what we want to do is raise the flag. So seen end of input equals true. And if it is not if it is not negative one only then we want to accumulate it into sum and number of students. So sum plus equals our current input and number of students should just increase. So taking a second look here we first initialize our accumulating variables sum and number of students. We initialize our flag to be down seen end of input false. And while our flag is down is basically read an input figure out if we want to raise the flag if it is negative one. Otherwise we accumulate the current input we add it to sum we count it as one of our valid grades. We increment number of students. So as long as our current is a grade we just accumulate it accumulate it and accumulate it. Once our current is negative one we raise the flag the Boolean condition is then checked and that makes this break out of the loop. After we do that basically we just calculate the average so just as before let's declare a double average variable and let's assign average to let's cast sum and divide it by the number of students. Once again let's cast number of students so then we get the average. Now we can announce the class average is and the value of the average variable and break the line. Let's execute it see that we didn't make any silly mistakes here ok. Enter the grades separated by space and your sequence by typing negative one. So let's say 90 87 76 89 and -1. And then the class average is 85.5 which is good. You can see that we didn't know how many numbers there were going to be. That made us choose a while loop here. That's typical when we don't know how many how long or how big is the how many times we want to repeat. That's usually when we pick when we choose a while statement. When we know the range of number of iterations we want to do then we choose a for loop just as we had in the counting program. It was a more readable more clear when we use the for loop. It was also in the first version of calculating the average when we knew we wanted to repeat number of students iterations we also chose a for loop. But when we don't know how many iterations we want to have a while loop is a better choice that what happened here. That was what happened when we calculated the sum of the digits of the number. We didn't know how many digits there were going to be so we used a while loop as long as we still have digits we want to keep on extracting the digits and adding it. So again as a rule of thumb here when you know the number of iterations you typically choose for loop. When you don't know it is more common to use a while loop.