CS 241 — Introduction to Problem Solving and Programming

Fundamentals of Programming

Flow of control, part III : More loop constructs

Jan 28, 2005

Overview

- Review: do-while and its deficiency
- While loops
- For loops as counting loops
- For loops as test-in-the-middle

Do-while loops

```
public class GoofyDoWhile {
    public static void main(String[] args) {
        int alohas =
            DocsIO.readint("How many times would you like me to say Aloha? ");

        do {
            System.out.println("Aloha.");
            alohas--;
        } while (alohas > 0);
    }
}
```

Do-while

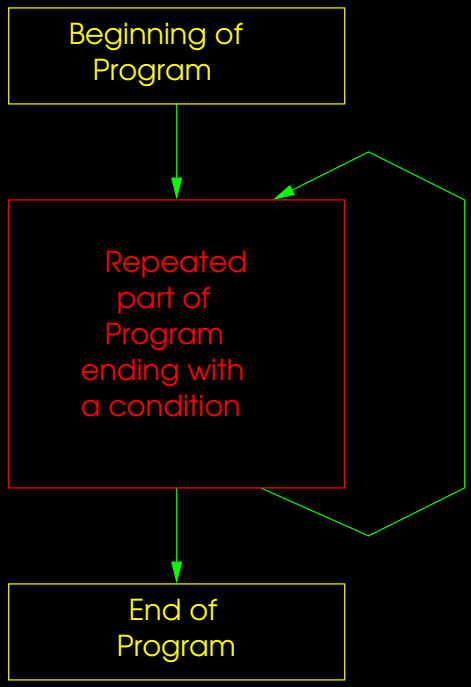
What would happen if we asked for no alohas?

```
How many times would you like me to say "Aloha"? 0
Aloha.
```

It always prints at least one aloha, no matter what.

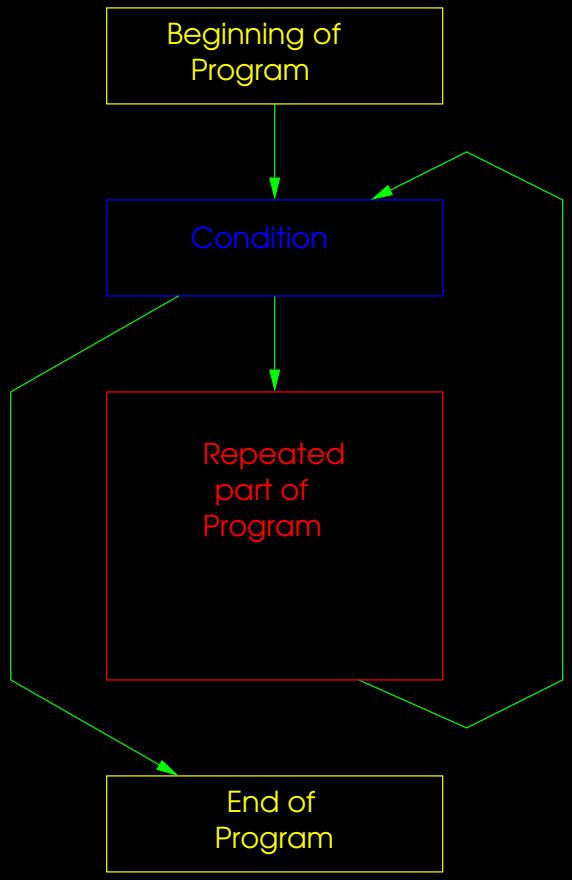
Loops

Do-while loops are called one-trip loops because they will have at least one iteration.
They are also called posttest loops because they test for continuation after an iteration.



While loops

Sometimes we want zero-trip loops, where the test can fail the first time.
They are also called pretest loops because they test for continuation before an iteration.



While loops

In Java, a loop of this kind is made with a while statement.

While statement: `while` (*BooleanExpression*)
Statement

Note that there is no separate semi-colon. The statement itself should end with a semi-colon; if the statement is a block, no semi-colon is needed after the closing curly brace.

Example

Our simple example revisited

```
public class GoofyWhile {
    public static void main(String[] args) {
        int alohas =
            DocsIO.readint("How many times would you like me to say \"Aloha\"? ");

        while (alohas > 0) {
            System.out.println("Aloha.");
            alohas--;
        }
    }
}
```

Example

```
ar1121: {13} java GoofyWhile
How many times would you like me to say "Aloha"? 0
ar1121: {14} java GoofyWhile
How many times would you like me to say "Aloha"? 5
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
```

Example

```
public class GoofyWhile {
    public static void main(String[] args) {
        boolean sayAgain = true;
        while (sayAgain) {
            System.out.println("Aloha.");
            char query = DocsIO.readchar("Do you want me to say it again (y/n)? ");
            if (query == 'n' || query == 'N')
                sayAgain = false;
        }
    }
}
```

Revised example, this time we query the user as we go along.

Example

```
public class GoofyWhile {
    public static void main(String[] args) {
        boolean sayAgain = true;
        while (sayAgain) {
            System.out.println("Aloha.");
            char query = DocsIO.readchar("Do you want me to say it again (y/n)? ");
            if (query == 'n' || query == 'N')
                sayAgain = false;
        }
    }
}
```

Notice we tell the user what sort of input we're expecting.

Example

```
public class GoofyWhile {
    public static void main(String[] args) {
        boolean sayAgain = true;
        while (sayAgain) {
            System.out.println("Aloha.");
            char query = DocsIO.readchar("Do you want me to say it again (y/n)? ");
            if (query == 'n' || query == 'N')
                sayAgain = false;
        }
    }
}
```

Notice that we allow for both lower case and capital N, but any erroneous input is assumed to be a "yes."

Example

```
ar1121: {21} java GoofyWhile
Aloha.
Do you want me to say it again (y/n)? y
Aloha.
Do you want me to say it again (y/n)? Y
Aloha.
Do you want me to say it again (y/n)? 4
Aloha.
Do you want me to say it again (y/n)? n
```

Finding an average

```
int total = 0;
int number = 0;
boolean moreVals = true;
while (moreVals) {
    total += DocsIO.readint("Please enter next value--> ");
    number++;
    char query = DocsIO.readchar("Do you have more  (y/n)? ");
    if (query == 'n' || query == 'N')
        moreVals = false;
}
System.out.println("Average: " + ((double) total / number));
```

A slightly more realistic program.

Finding an average

```
int total = 0;
int number = 0;
boolean moreVals = true;
while (moreVals) {
    total += DocsIO.readint("Please enter next value--> ");
    number++;
    char query = DocsIO.readchar("Do you have more  (y/n)? ");
    if (query == 'n' || query == 'N')
        moreVals = false;
}
System.out.println("Average: " + ((double) total / number));
```

This is a little tricky. We avoid an extra variable.

Finding an average

```
ar1121: {27} java WhileAverage
Please enter next value--> 14
Do you have more  (y/n)? y
Please enter next value--> 64
Do you have more  (y/n)? y
Please enter next value--> 22
Do you have more  (y/n)? y
Please enter next value--> 81
Do you have more  (y/n)? n
Average: 45.25
```

Counting letter occurrences

Let's try a non-trivial example.

> Write a program that asks the user to enter a string and a character and then calculates the number of times that character occurs in the string.

How would we do it?

- What would each iteration of the loop do?
- How would we know when we're finished?
- What variables would be the same throughout, which would change per iteration, and which would exist only on a single iteration?

Counting letter occurrences

- Each iteration would inspect a specific character of the string (from first to last).
- We're finished when we have looked at all the characters (or, reached the end of the loop).
- "Constant" variables:
  - The user-given string.
  - The user-given character.
- Loop variables:
  - How many occurrences we have seen so far.
  - Our current position in the string.
- Iteration variable:
  - The current character

Algorithm

- Read `text` from the user.
- Read `searchItem` from the user.

Algorithm

- Read `text` from the user.
- Read `searchItem` from the user.
- Initialize `position` to zero.

Algorithm

- Read `text` from the user.
- Read `searchItem` from the user.
- Initialize `position` to zero.
- Initialize `occurrences` to zero.

Algorithm

- Read `text` from the user.
- Read `searchItem` from the user.
- Initialize `position` to zero.
- Initialize `occurrences` to zero.
- While `position` is less than the length of `text`,
  - Pick `current` character from `position` in `text`.

Algorithm

- Read `text` from the user.
- Read `searchItem` from the user.
- Initialize `position` to zero.
- Initialize `occurrences` to zero.
- While `position` is less than the length of `text`,
  - Pick `current` character from `position` in `text`.
  - If `current` equals `searchItem`, increment `occurrences`

Algorithm

- Read `text` from the user.
- Read `searchItem` from the user.
- Initialize `position` to zero.
- Initialize `occurrences` to zero.
- While `position` is less than the length of `text`,
  - Pick `current` character from `position` in `text`.
  - If `current` equals `searchItem`, increment `occurrences`
  - Increment `position`

Algorithm

- Read `text` from the user.
- Read `searchItem` from the user.
- Initialize `position` to zero.
- Initialize `occurrences` to zero.
- While `position` is less than the length of `text`,
  - Pick `current` character from `position` in `text`.
  - If `current` equals `searchItem`, increment `occurrences`
  - Increment `position`
- Display `occurrences`

Code

```
String text = DocsIO.readString("Please enter the string--> ");
char searchItem = DocsIO.readchar("Please enter the search item--> ");
int occurrences = 0;
int position = 0;

while(position < text.length()) {
    char current = text.charAt(position);
    if (current == searchItem) occurrences++;
    position++;
}
System.out.println(occurrences + " occurrences found.");
```

Code

```
String text = DocsIO.readString("Please enter the string--> ");
char searchItem = DocsIO.readchar("Please enter the search item--> ");
int occurrences = 0;
int position = 0;

while(position < text.length()) {
    char current = text.charAt(position);
    if (current == searchItem) occurrences++;
    position++;
}
System.out.println(occurrences + " occurrences found.");
```

Whole-program variables which we don't change after initialization.

Code

```
String text = DocsIO.readString("Please enter the string--> ");
char searchItem = DocsIO.readchar("Please enter the search item--> ");
int occurrences = 0;
int position = 0;

while(position < text.length()) {
    char current = text.charAt(position);
    if (current == searchItem) occurrences++;
    position++;
}
System.out.println(occurrences + " occurrences found.");
```

A variable which the loop updates because it is the goal of our program.

Code

```
String text = DocsIO.readString("Please enter the string--> ");
char searchItem = DocsIO.readchar("Please enter the search item--> ");
int occurrences = 0;
int position = 0;

while(position < text.length()) {
    char current = text.charAt(position);
    if (current == searchItem) occurrences++;
    position++;
}
System.out.println(occurrences + " occurrences found.");
```

A variable marking our current problem piece and when we are done.

It is a loop counter and an index into `text`.

Code

```
String text = DocsIO.readString("Please enter the string--> ");
char searchItem = DocsIO.readchar("Please enter the search item--> ");
int occurrences = 0;
int position = 0;

while(position < text.length()) {
    char current = text.charAt(position);
    if (current == searchItem) occurrences++;
    position++;
}
System.out.println(occurrences + " occurrences found.");
```

A variable coming to life and dying each iteration.

Running

```
Please enter the string--> Arma virumque cano Troiae qui primus ob oris
Please enter the search item--> e
2 occurrences found.
ar1121: {38} java CharCounter
Please enter the string--> Arma virumque cano Troiae qui primus ob oris
Please enter the search item--> o
4 occurrences found.
```

Adjustable average

Going back to our average example. . .

It's rather inconvenient to prompt the user each time if there are more values. Usually the user knows the number ahead of time.

So, *prompt* the user, and use the result to determine the number of loop iterations.

Adjustable average

```
public class WhileAverage {
    public static void main(String[] args) {
        int total = 0;
        int number = DocsIO.readint("How many values do you want to average? " );









    }
}
```

Adjustable average

```
public class WhileAverage {
    public static void main(String[] args) {
        int total = 0;
        int number = DocsIO.readint("How many values do you want to average? " );
        int current = 0;









    }
}
```

Adjustable average

```
public class WhileAverage {
    public static void main(String[] args) {
        int total = 0;
        int number = DocsIO.readint("How many values do you want to average? " );
        int current = 0;
        while (current < number) {


        }

    }
}
```

Adjustable average

```
public class WhileAverage {
    public static void main(String[] args) {
        int total = 0;
        int number = DocsIO.readint("How many values do you want to average? " );
        int current = 0;
        while (current < number) {
            total += DocsIO.readint("Please enter next value--> ");

        }

    }
}
```

Adjustable average

```
public class WhileAverage {
    public static void main(String[] args) {
        int total = 0;
        int number = DocsIO.readint("How many values do you want to average? " );
        int current = 0;
        while (current < number) {
            total += DocsIO.readint("Please enter next value--> ");
            current++;
        }

    }
}
```

Adjustable average

```
public class WhileAverage {
    public static void main(String[] args) {
        int total = 0;
        int number = DocsIO.readint("How many values do you want to average? " );
        int current = 0;
        while (current < number) {
            total += DocsIO.readint("Please enter next value--> ");
            current++;
        }
        System.out.println("Average: " + ((double) total / number));
    }
}
```

Adjustable average

```
How many values do you want to average? 6
Please enter next value--> 3
Please enter next value--> 4
Please enter next value--> 5
Please enter next value--> 6
Please enter next value--> 7
Please enter next value--> 8
Average: 5.5
```

Pattern

```
public class WhileAverage {
    public static void main(String[] args) {
        int total = 0;
        int number = DocsIO.readint("How many values do you want to average? " );
        int current = 0;
        while (current < number) {
            total += DocsIO.readint("Please enter next value--> ");
            current++;
        }
        System.out.println("Average: " + ((double) total / number));
    }
}
```

Can anyone detect a pattern?

Pattern

```
public class WhileAverage {
    public static void main(String[] args) {
        int total = 0;
        int number = DocsIO.readint("How many values do you want to average? " );
        int current = 0;
        while (current < number) {
            total += DocsIO.readint("Please enter next value--> ");
            current++;
        }
        System.out.println("Average: " + ((double) total / number));
    }
}
```

Initialize, test, increment.

Pattern

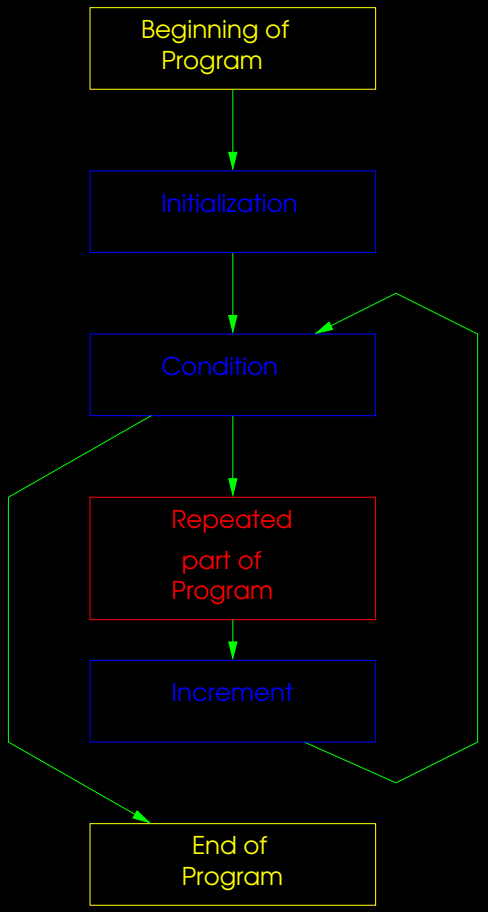
```
String text = DocsIO.readString("Please enter the string--> ");
char searchItem = DocsIO.readchar("Please enter the search item--> ");
int occurrences = 0;
int position = 0;

while(position < text.length()) {
    char current = text.charAt(position);
    if (current == searchItem) occurrences++;
    position++;
}
System.out.println(occurrences + " occurrences found.");
```

Initialize, test, increment.

For loops

Loops in this pattern are called counting loops. When patterns are this common, they often become part of the language.



While loops

In Java, a loop of this kind is made with a for statement.

For statement: `for` (*VariableDeclaration*; *BooleanExpresion*; *Expression*)
Statement

Equivalent to:

VariableDeclaration
`while` (*BooleanExpression*) {
Statements
Experssion;
}

Example

```
int total = 0;
int number = DocsIO.readint("How many values do you want to average? " );

for (int i = 0; i < number; i++)
    total += DocsIO.readint("Please enter next value--> ");

System.out.println("Average: " + ((double) total / number));
```

Traditionally, loop counters are named `i`, after the common use in mathematics (also, first letter of "index" and is almost always an `int`).

This is one of the few cases where single-variable letters are considered good style.

Another try

How would we print out each character in a string, one line at a time?

```
String text = DocsIO.readString("Please enter the string--> ");

for (                                  )
    System.out.println(text.charAt(i));
```

Another try

How would we print out each character in a string, one line at a time?

```
String text = DocsIO.readString("Please enter the string--> ");

for (int i = 0;                          )
    System.out.println(text.charAt(i));
```

Another try

How would we print out each character in a string, one line at a time?

```
String text = DocsIO.readString("Please enter the string--> ");

for (int i = 0; i < text.length();    )
    System.out.println(text.charAt(i));
```

Another try

How would we print out each character in a string, one line at a time?

```
String text = DocsIO.readString("Please enter the string--> ");

for (int i = 0; i < text.length(); i++)
    System.out.println(text.charAt(i));
```

More tricky

How about *backwards*?

```
String text = DocsIO.readString("Please enter the string--> ");

for (                                    )
    System.out.println(text.charAt(i));
```

More tricky

How about *backwards*?

```
String text = DocsIO.readString("Please enter the string--> ");

for (int i = text.length() - 1;            )
    System.out.println(text.charAt(i));
```

More tricky

How about *backwards*?

```
String text = DocsIO.readString("Please enter the string--> ");

for (int i = text.length() - 1; i >= 0;    )
    System.out.println(text.charAt(i));
```

More tricky

How about *backwards*?

```
String text = DocsIO.readString("Please enter the string--> ");

for (int i = text.length() - 1; i >= 0; i--)
    System.out.println(text.charAt(i));
```

Prime factorization

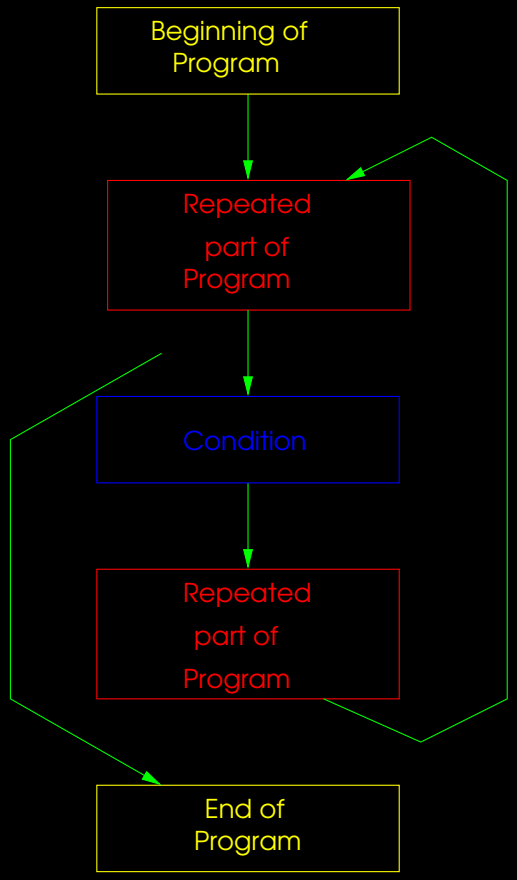
Suppose we want average a series of numbers, but instead of making the user commit to a number before hand, wait until the user enters a sentinel value (say, -1 to indicate the end of the list).

An algorithm:

- Initialize `total` to zero.
- Initialize `number` to zero.
- Repeat
  - Input the next number, `current`, from the user.
  - If `current` is -1, then quit.
  - Add `current` to `total`
  - Increment `number`
- If `number` is not zero, display `total` / `number`

Forever loops

The structure would look like this.
Such a loop is called a test-in-the-middle loop or a forever loop or a loop and a half.



Forever loops

This can be implemented with a for loop. For one thing, you could put bogus expressions in the header.

But actually, you can leave them blank. The for header expressions are optional. (Use all, any, or none.)

```
for ( ; ; )
```

Use a `break` statement to quit.

Forever loops

```
int total = 0;
int number = 0;

for (;;) {
    int current =  DocsIO.readint("Please enter next value (-1 to quit)-->
    if (current == -1)
        break;
    total += current;
    number++;
}
if (number != 0)
    System.out.println("Average: " + ((double) total / number));
```

Forever loops

```
Please enter next value (-1 to quit)--> 6
Please enter next value (-1 to quit)--> 7
Please enter next value (-1 to quit)--> 8
Please enter next value (-1 to quit)--> 9
Please enter next value (-1 to quit)--> 1
Please enter next value (-1 to quit)--> 2
Please enter next value (-1 to quit)--> 3
Please enter next value (-1 to quit)--> -1
Average: 5.142857142857143
```

Summary

Be able to recognized the following terms.

- One-trip / posttest loop
- Zero-trip / pretest
- `while` statement
- Loop counter / index
- Counting loop
- `for` statement
- Sentinel
- Test-in-the-middle loop / forever loop / loop and a half
- `break` statement