CS 241 — Introduction to Problem Solving and Programming

Fundamentals of Programming

Flow of control, part II : Loop constructs

Jan 26, 2005

From last time: Comparing Strings

```
String name = DocsIO.readString("Please enter your name--> ");
String job = DocsIO.readString("Hello, " + name + ", what is your occupation? ");

if (name.equals("Bill"))
    System.out.println("What a coincidence, " + name +
                       "! My uncle is also named Bill and works at  " + job + ".");
else if (job.equals("writing novels"))
    System.out.println("I never imagined a novelist would be named " + name);
else
    System.out.println(name + " is a funny name, but I respect the career of "
                       + job);
```

`equals(String)` is a `String` method that returns a `boolean`, `true` if the `Strings` contain the same sequence of characters.

From last time: Comparing Strings

```
String name = DocsIO.readString("Please enter your name--> ");
String job = DocsIO.readString("Hello, " + name + ", what is your occupation? ");

if (name.equals("Bill"))
    System.out.println("What a coincidence, " + name +
                       "! My uncle is also named Bill and works at  " + job + ".");
else if (job.equals("writing novels"))
    System.out.println("I never imagined a novelist would be named " + name);
else
    System.out.println(name + " is a funny name, but I respect the career of "
                       + job);
```

Like other `DocsIO` methods we've seen, `DocsIO.readString` prompts the user, but returns a `String`, the entire line the user enters.

From last time: Comparing Strings

```
String name = DocsIO.readString("Please enter your name--> ");
String job = DocsIO.readString("Hello, " + name + ", what is your occupation? ");

if (name.equals("Bill"))
    System.out.println("What a coincidence, " + name +
                       "! My uncle is also named Bill and works at  " + job + ".");
else if (job.equals("writing novels"))
    System.out.println("I never imagined a novelist would be named " + name);
else
    System.out.println(name + " is a funny name, but I respect the career of "
                       + job);
```

```
...

Please enter your name--> Bill
Hello, Bill, what is your occupation? writing novels
What a coincidence, Bill! My uncle is also named Bill and works at  writing novels.
```

From last time: Comparing Strings

```
        String name = DocsIO.readString("Please enter your name--> ");
        String job = DocsIO.readString("Hello, " + name + ", what is your occupation? ");

        if (name.equals("Bill"))
            System.out.println("What a coincidence, " + name +
                               "! My uncle is also named Bill and works at  " + job + ".");
        else if (job.equals("writing novels"))
            System.out.println("I never imagined a novelist would be named " + name);
        else
            System.out.println(name + " is a funny name, but I respect the career of "
                               + job);

...

Please enter your name--> Aggripinilla
Hello, Aggripinilla, what is your occupation? writing novels
I never imagined a novelist would be named Aggripinilla
```

From last time: Comparing Strings

```
String name = DocsIO.readString("Please enter your name--> ");
String job = DocsIO.readString("Hello, " + name + ", what is your occupation? ");

if (name.equals("Bill"))
    System.out.println("What a coincidence, " + name +
                       "! My uncle is also named Bill and works at  " + job + ".");
else if (job.equals("writing novels"))
    System.out.println("I never imagined a novelist would be named " + name);
else
    System.out.println(name + " is a funny name, but I respect the career of "
                       + job);
```

...

```
Please enter your name--> Vercingetorix
Hello, Vercingetorix, what is your occupation? fighting Romans
Vercingetorix is a funny name, but I respect the career of fighting Romans
```

From last time: Warnings about branches

- Don't mix up = and ==.
- Watch out for missing {.
- Declare variables in the right place.
- Be careful about initializing variables in branches.
- Make sure your `if`s and `else`s match as you expect.
- Remember `&&` has higher precedence than || (or just use parentheses).
- Use `.equals()` to compare `String`s.

From last time: Summary

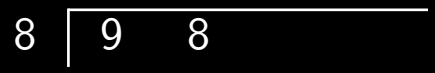
Be able to identify the following concepts:

- `boolean` type
- Lexicographical order
- Branch statement
- Flow of control
- Block statement
- Scope
- Multibranch
- Short-circuit evaluation

Overview

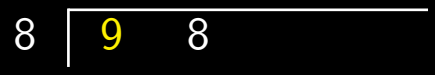
- General need for a loop
- Do-While loops
- Extended example

Problem



Let's begin with a problem . . .
long division.
What is the first step?

Problem

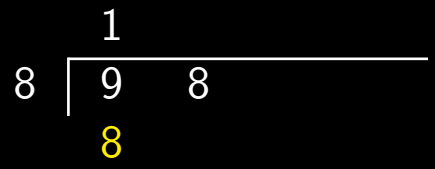


First, we chip off a piece of the problem, something which is less than ten times the divisor.

Long division

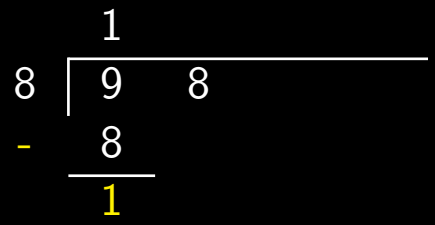
Then we do simple, integer division.

Long division



We multiply that result by the divisor.

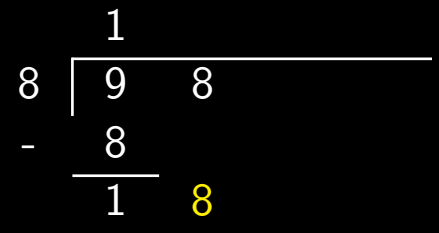
Long division



And subtract that result from the current piece of the problem.

Long division

We drop the next digit, which with the previous result become the new piece of the problem.



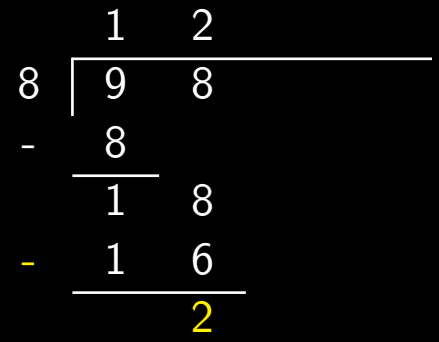
Long division

Divide and multiply.

```
    1  2
  ------
8 | 9  8
- | 8
  ---
    1  8
    1  6
```

Long division

Subtract.



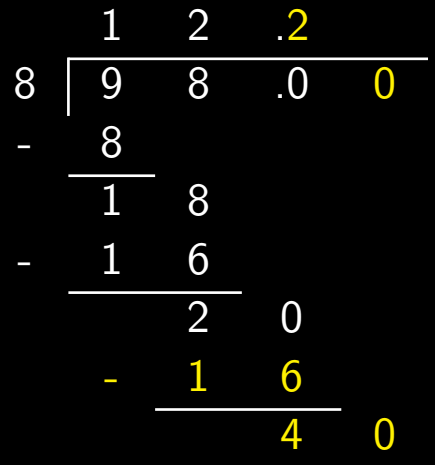
Long division

This time, in order to drop, we must also add a decimal point and a zero.

```
    1  2  .
  ------------
8 | 9  8  .0
- | 8
  ---
    1  8
- | 1  6
  ------
       2  0
```

Long division

Divide, multiply, subtract, and drop.



Long division

Divide, multiply, subtract.

```
      1   2   .2   5
   ---------------------
8 |  9   8   .0   0
  -  8
  ---
     1   8
   -  1   6
   ---------
         2   0
       -  1   6
       ---------
             4   0
           -  4   0
           ---------
                 0
```

Long division

We're finished, because the current piece of the problem is zero, and there are no more non-decimal places to drop.

```
      1   2   .2   5
   ---------------
8 | 9   8   .0   0
  - 8
  ---
    1   8
  - 1   6
  -------
        2   0
      - 1   6
      -------
            4   0
          - 4   0
          -------
                0
```

Long division

What's our algorithm?

- Pick off the next digit of the dividend, add to the end of the current problem piece.

Long division

What's our algorithm?

- Pick off the next digit of the dividend, add to the end of the current problem piece.
- (Integer) divide current piece by divisor, call it *quotient*.

Long division

What's our algorithm?

- Pick off the next digit of the dividend, add to the end of the current problem piece.
- (Integer) divide current piece by divisor, call it *quotient*.
- Add quotient to result.

Long division

What's our algorithm?

- Pick off the next digit of the dividend, add to the end of the current problem piece.
- (Integer) divide current piece by divisor, call it *quotient*.
- Add quotient to result.
- Multiply quotient by divisor, call it *product*.

Long division

What's our algorithm?

- Pick off the next digit of the dividend, add to the end of the current problem piece.
- (Integer) divide current piece by divisor, call it *quotient*.
- Add quotient to result.
- Multiply quotient by divisor, call it *product*.
- Subtract current piece by product, make it the new current piece.

Long division

What's our algorithm?

- Pick off the next digit of the dividend, add to the end of the current problem piece.
- (Integer) divide current piece by divisor, call it *quotient*.
- Add quotient to result.
- Multiply quotient by divisor, call it *product*.
- Subtract current piece by product, make it the new current piece.
- Repeat until current is zero and there are no more original digits to drop.

Long division

What's our algorithm?

- Initialize current piece to zero.
- Do
  - Pick off the next digit of the dividend, add to the end of the current problem piece.
  - (Integer) divide current piece by divisor, call it *quotient*.
  - Add quotient to result.
  - Multiply quotient by divisor, call it *product*.
  - Subtract current piece by product, make it the new current piece.

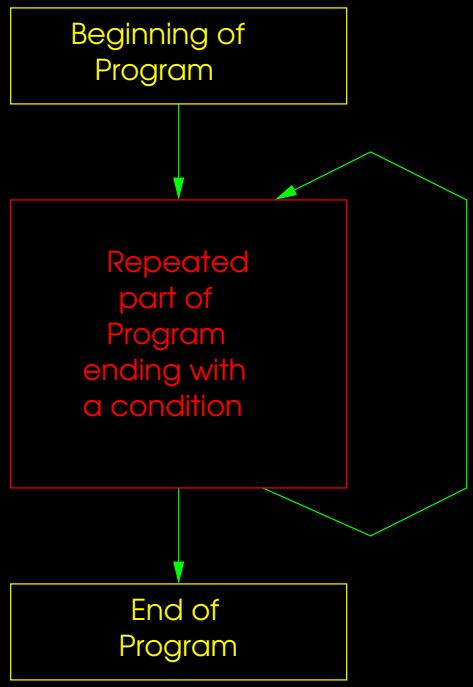
  until current is zero and there are no more original digits to drop.
- Display result.

Loops

We have seen branches in the flow of control.

Now we want not just to split the flow, but for the flow to go back to an earlier program point. We want repetition.

A repetition structure in a program is called a loop.



Do-while loops

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

Do-while statement: `do`
Statement
`while (`*BooleanExpression*`);`

Note the semi-colon at the end.

The inside, repeated statement is called the body. The boolean expression is the test or condition of the loop. Each trip through the loop is called an iteration.

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);
    }
}
```

Simple example.

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);
    }
}
```

Initialize a variable to stand for "number of alohas left to say."

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);
    }
}
```

The loop.

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);
    }
}
```

Keep repeating "while there are still some alohas left" (or "until there are no alohas left").

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);
    }
}
```

For each iteration, print an "aloha."

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);
    }
}
```

. . . and make sure you adjust the counting variable or counter. The condition depends on it.

Do-while loops

```
ar1121: {178} java GoofyDoWhile
How many times would you like me to say "Aloha"? 1
Aloha.
ar1121: {179} java GoofyDoWhile
How many times would you like me to say "Aloha"? 3
Aloha.
Aloha.
Aloha.
ar1121: {180} java GoofyDoWhile
How many times would you like me to say "Aloha"? 8
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
```

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);
    }
}
```

What if we left out decrementing alohas?

Do-while loops

```
How many times would you like me to say "Aloha"? 1
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
Aloha.
```

Long division

Returning to our long division example. . .

- Initialize current piece to zero.
- Do
  - Pick off the next digit of the dividend, add to the end of the current problem piece.
  - (Integer) divide current piece by divisor, call it *quotient*.
  - Add quotient to result.
  - Multiply quotient by divisor, call it *product*.
  - Subtract current piece by product, make it the new current piece.

  until current is zero and there are no more original digits to drop.
- Display result.

Long division

```
String dividend = DocsIO.readString("Please enter the dividend--> ");
int divisor = DocsIO.readint("Please enter the divisor--> ");

int position = 0;         // The position of the current digit in dividend
int current = 0;         // The current piece of the problem
String result = "";      // The quotient for the entire long division so far

do {
     ...
```

We'll store the dividend as a `String` and pick off digits using `substring`.

Long division

```
String dividend = DocsIO.readString("Please enter the dividend--> ");
int divisor = DocsIO.readint("Please enter the divisor--> ");

int position = 0;         // The position of the current digit in dividend
int current = 0;         // The current piece of the problem
String result = "";      // The quotient for the entire long division so far

do {
     ...
```

Three variables keep track of values modified by the loop. We initialize them for starting (initial) values.

Long division

```
do {
    current *= 10;
    if (position < dividend.length())
        current += Integer.parseInt(dividend.substring(position, position+1));
    else if (position == dividend.length())
        result += ".";

    int quotient = current / divisor;
    result += quotient;
    int product = quotient * divisor;
    current -= product;
    position++;
} while(current != 0 || position < dividend.length());
```

First step: make room for the dropped value.

Long division

```
do {
    current *= 10;
    if (position < dividend.length())
        current += Integer.parseInt(dividend.substring(position, position+1));
    else if (position == dividend.length())
        result += ".";

    int quotient = current / divisor;
    result += quotient;
    int product = quotient * divisor;
    current -= product;
    position++;
} while(current != 0 || position < dividend.length());
```

If we haven't hit the decimal point yet, take the next integer from the dividend, and "drop" it.

Long division

```
do {
    current *= 10;
    if (position < dividend.length())
        current += Integer.parseInt(dividend.substring(position, position+1));
    else if (position == dividend.length())
        result += ".";

    int quotient = current / divisor;
    result += quotient;
    int product = quotient * divisor;
    current -= product;
    position++;
} while(current != 0 || position < dividend.length());
```

Special case: If this is the first time we've gotten past the end of the dividend, add a decimal point to the remainder.

Long division

```
do {
    current *= 10;
    if (position < dividend.length())
        current += Integer.parseInt(dividend.substring(position, position+1));
    else if (position == dividend.length())
        result += ".";

    int quotient = current / divisor;
    result += quotient;
    int product = quotient * divisor;
    current -= product;
    position++;
} while(current != 0 || position < dividend.length());
```

Divide step.

Long division

```
do {
    current *= 10;
    if (position < dividend.length())
        current += Integer.parseInt(dividend.substring(position, position+1));
    else if (position == dividend.length())
        result += ".";

    int quotient = current / divisor;
    result += quotient;
    int product = quotient * divisor;
    current -= product;
    position++;
} while(current != 0 || position < dividend.length());
```

String concatenation. Add quotient (automatically cast to a `String`) to to our result `String`.

Long division

```
do {
    current *= 10;
    if (position < dividend.length())
        current += Integer.parseInt(dividend.substring(position, position+1));
    else if (position == dividend.length())
        result += ".";

    int quotient = current / divisor;
    result += quotient;
    int product = quotient * divisor;
    current -= product;
    position++;
} while(current != 0 || position < dividend.length());
```

Multiply step.

Long division

```
do {
    current *= 10;
    if (position < dividend.length())
        current += Integer.parseInt(dividend.substring(position, position+1));
    else if (position == dividend.length())
        result += ".";

    int quotient = current / divisor;
    result += quotient;
    int product = quotient * divisor;
    current -= product;
    position++;
} while(current != 0 || position < dividend.length());
```

Update the current problem piece and increment the position.

Long division

```
do {
    current *= 10;
    if (position < dividend.length())
        current += Integer.parseInt(dividend.substring(position, position+1));
    else if (position == dividend.length())
        result += ".";

    int quotient = current / divisor;
    result += quotient;
    int product = quotient * divisor;
    current -= product;
    position++;
} while(current != 0 || position < dividend.length());
```

Check if we are ready to quit.

Long division

```
    ...
} while(current != 0 || position < dividend.length());

System.out.println("Result: " + result);
```

Print the result.

Long division

```
    ...
} while(current != 0 || position < dividend.length());

System.out.println("Result: " + result);
```

If this condition never fails, the program will never end.

We would have an infinite loop.

Preview

Going back to our "aloha" example. . .

What would happen if we asked for no alohas?

Preview

Going back to our “aloha” example. . .

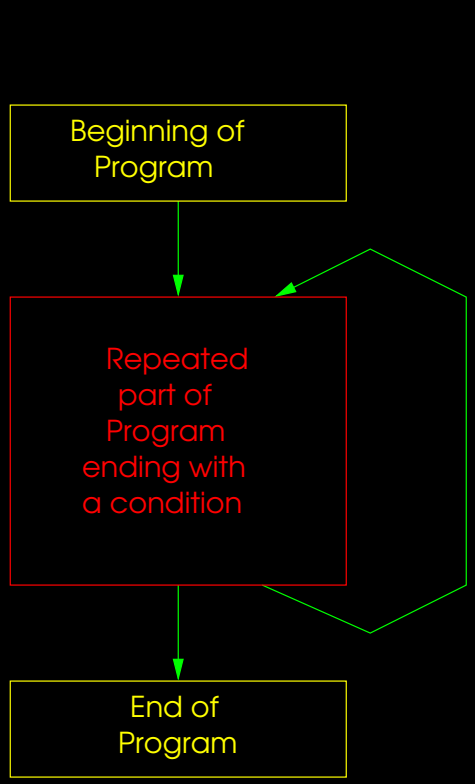
What would happen if we asked for no alohas?

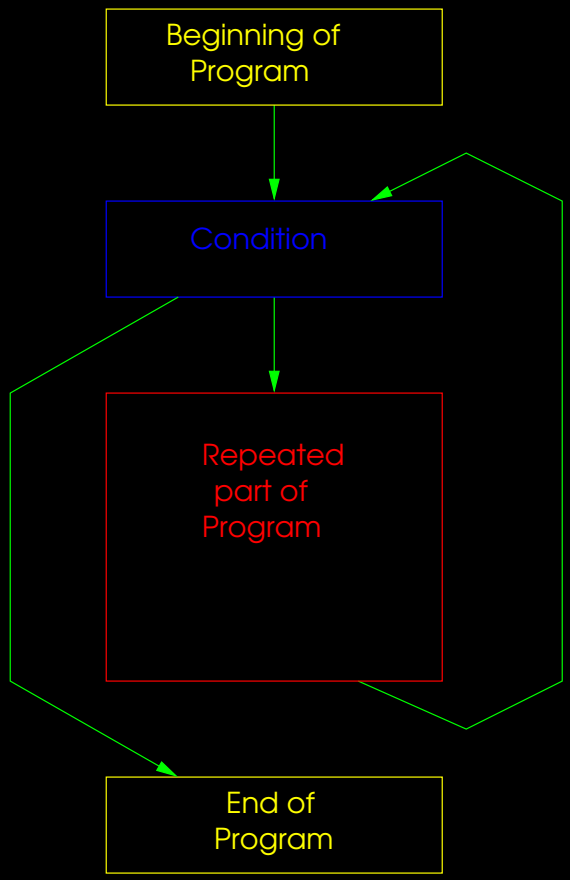
```
How many times would you like me to say "Aloha"? 0
Aloha.
```

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

Preview

What we want is to be able to test at the beginning, not the end.





Summary

Be able to identify the following concepts:

- Loop
- Body
- Iteration
- Counter
- Infinite loop