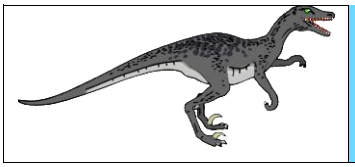
Comp 1421 Foundations of Computing

Workshop3 Control Flow and Loops

CBPrice September 2019

Color Coding for this Worksheet	
	Information
	Instructed Learning
	Autonomous Learning “Homework”
	Assignment material
	Research



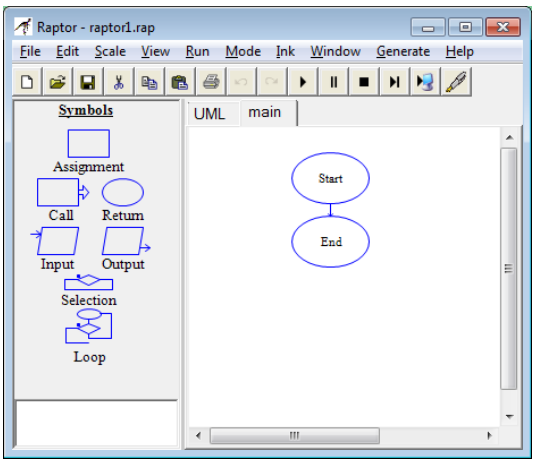
The Raptor flow diagram tool is free for download at the following link
https://raptor.martincarlisle.com/

Activities

1 **Building a Loop in Raptor**

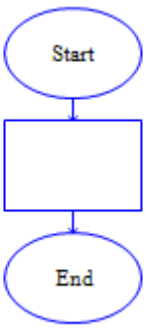
(a) Download the Raptor Files, unzip and place in a location of your choice.

(b) Open the file raptor1.rap where you should see the following



(c) Make sure the Mode is set to “Intermediate”.

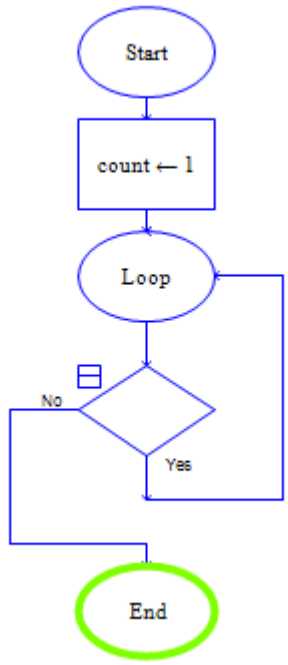
(d) Now left-click on the assignment box and drag into place like this



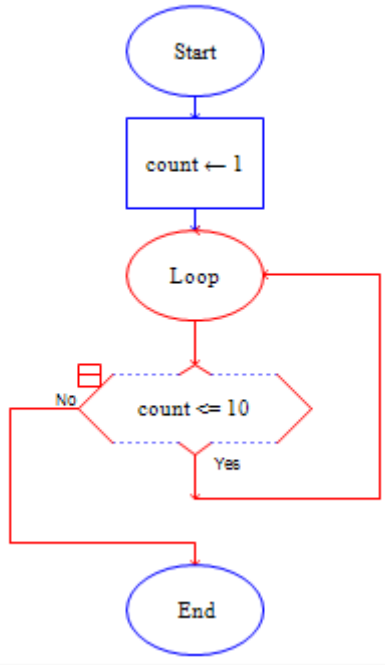
(e) Double left click on the assignment box and a dialog box will appear. In the field “Set” enter the variable name **count**. Then in the field “to” enter the value **1** which will assign **1** to the variable **count.**

(f) Run the flow diagram by clicking on the  icon, clicking several times to step through the diagram. You should see the value of count being assigned in the bottom left window.

(g) Now add the loop construct and drag into place like this



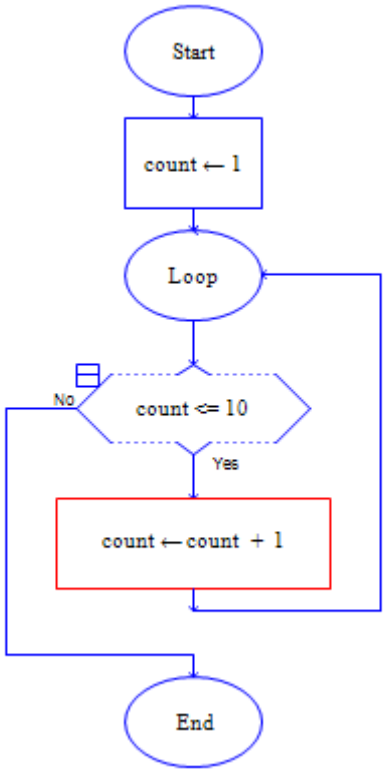
(h) Double-left click on the loop construct and in the "Enter loop condition" box, type **count <= 10** and you should see this



(i) Finally drag another assignment box in the location shown below, then double-left click on it and in the field "Set" type **count** and in the field "to" type **count + 1**.

(j) Run the flow diagram. Keep an eye on the value of count shown in red, bottom left. Answer these questions

1) How many times does the line **count <- count + 1** get executed?
2) How many times does the condition **if count <= 10** get executed?
3) Why are these not the same?



(k) Stop the flow diagram running. Select "Generate" from the toolbar and select "Java". Look in your folder and you will see a file **raptor1.java** Open this with Notepad++ and you'll recognise the code associated with the flow diagram. Sweet eh?

2 **Using a Loop to sum the first 10 digits**

Here you are going to add two extra boxes, so when the loop runs, it will calculate 1 + 2 + 3 + … + 9 + 10. Let's agree to use the variable **sum**.

a) You will need one assignment statement to initialize **sum** to **0**. Think where this box should go in the flow diagram and put it there.

b) You will need an assignment statement to calculate the running sum. This will represent the following: **sum = sum + count**. Think where this assignment statement should go and put it there

c) Finally we'll need to output the result. Drag the output symbol to the appropriate place in the flow diagram (where the sum is complete). Write the following in the output box: **"Sum = " + sum**

d) Now run the flow diagram and check that it outputs the correct value. But what should it be? Well, we can re-arrange 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 like this
(1 + 9) + (2 + 8) + (3 + 7) + … you get the idea. Continue with this rearrangement and calculate what the sum should be.

3 **Summing different numbers of digits, using a parameter**

In the above example we summed numbers from 1 to 10. Let's say we want to sum to any desired

number, perhaps 11, 13, 55 or 100. So our flow diagram needs to ask the user for some input, to say what number to sum to

(a) Add an input box to your flowchart at the appropriate point, open it up and type in the prompt "Input Final Number" (or some other meaningful prompt). Then in the variable box, type the variable name **finalNumber**.

(b) Now we have to change the loop condition, which is set to **count <= 10**. Think what this should be. OK, so here's confirmation of what you decided **count <= finalNumber.**

(c) Run the flow diagram and check that it works for a few values of **finalNumber**

The work so far could form part of your first assignment. So you may wish to capture the flow diagram and drop this into a Word document. The way to do this is File > Print to clipboard, then right-click in the Word document and choose paste.

4. **Summing Numbers in WeeBee code.**

Here we are going to directly code the flow diagram solution you worked on above. The idea is to be able to see a flow diagram and a coded solution side-by-side by Sondheim.

You should be familiar with loops in WeeBee code from the previous worksheet.

(a) Open up **activity5.cde** which is a template containing variable declarations and a skeleton loop. You need to complete the code. Let's forget about getting an input for the moment, and just sum the numbers 1 to 10.

i) Initialize the variable **sum** like you did above. Think where this line of code should go.
ii) Now add a line of code to update the value of sum, like you did above. Think where this should go
iii) To output the sum we can get Pip to *speak* the value of sum like this:

```
pip.says("Sum is " + sum);
pip.rest();
```

Now run the code.

(b) How can we grab user input using the WeeBee engine? Here's how, but take note, this line of code needs to be placed in the **once()** method.

```
finalNumber = pip.asksForNumInt("Input Final Number");
```

You will have to modify the loop condition, just as you did above.

(c) Run the code and see what happens. Maybe you will need to do some debugging.

You could use the results of activity 4 in your first assignment, alongside the flow diagram you created in activity 3.

5. **Storying or Commenting: Execution or Source.**

As explained in class, we are working together on a research project looking at "storying" code, getting actors to say what is happening. The classical way of documenting code is "commenting", in Java this is done using a double forward slash like this. Here's an example

```
// Update the count variable
count = count + 1;
```

This can also be used in the WeeBee engine. Such commenting is placed in the "source code" and serves as a reminder when you are reading the code. But it does not make its way to the screen when the code runs. The WeeBee engine can make this happen, and our research is to see if this helps reading and writing code. Here's the above example again

```
pip.says("I'm updating the count variable");
pip.rest();
count = count + 1;
```

(a) Open up **activity6.cde** which contains the complete code for summing. You will see lines which look like this //----------

(b) **Either** complete these, using in-line commenting, **or** get Pip to speak the comments.

6. **The "Notional Machine" – Control Flow**

Control Flow is all about how your algorithm, expressed as a flow diagram, or as Java code (WeeBee engine) decides what to do next. The work you have done so far has used two of the three elements of control flow: (i) sequence, (ii) iteration.

(a) Looking at your flow diagram you should see several examples of *sequence*. Look for two or more boxes following each other. Annotate your flow diagram (in your Word document) showing examples of *sequence.*

(b) Iteration. Well, there is clearly one while loop in your flow diagram. Here you should identify which *variables* are used to control the loop. **HINT:** The variable **sum** is not part of *control*; it simply makes the sum (that's called *data flow* which we shall explore later). So you could highlight the variables (in your WeeBee code, or your flow diagram) which are to do with control flow.

Some reflections about your understanding of "control flow" would be useful for the ass.

7. **Thinking Language**

Often when we think, we talk to ourselves, inside our heads. So it's important to get the language right, especially when it comes to reading and writing code.

Many computing languages use the equals sign (**=**) for an assignment, e.g., **variable = 10;** This can be confusing to some who interpret the equals sign as a *mathematical equality.* When you see a statement like **variable = 10;** then you should think "the variable **becomes** 10". So think "becomes" rather than "equals". There we go.