Loops

Today

- switch case
- Common errors with control structures
- While loop
- do While Loop

Due this week

- **Recitation 3**
- **Homework 3**
  - Write solutions in VSCode and paste in Autograder, **Homework 3 CodeRunner**.
  - Zip your .cpp files and submit on canvas **Homework 3**.

The `switch` Statement vs. the `if` statement

- Below is a complicated if() statement to choose a text string to assign based on the value of an int variable:

```
int digit;
…  //digit variable gets set here by some code
if (digit == 1) { digit_name = "one"; }
else if (digit == 2) { digit_name = "two"; }
else if (digit == 3) { digit_name = "three"; }
else if (digit == 4) { digit_name = "four"; }
else if (digit == 5) { digit_name = "five"; }
else if (digit == 6) { digit_name = "six"; }
else if (digit == 7) { digit_name = "seven"; }
else if (digit == 8) { digit_name = "eight"; }
else if (digit == 9) { digit_name = "nine"; }
else { digit_name = ""; }
```

The `switch` Statement

- The switch statement is an alternative to nested `if() else` statements. But `switch` is at least as awkward to code as nested `if() else`:

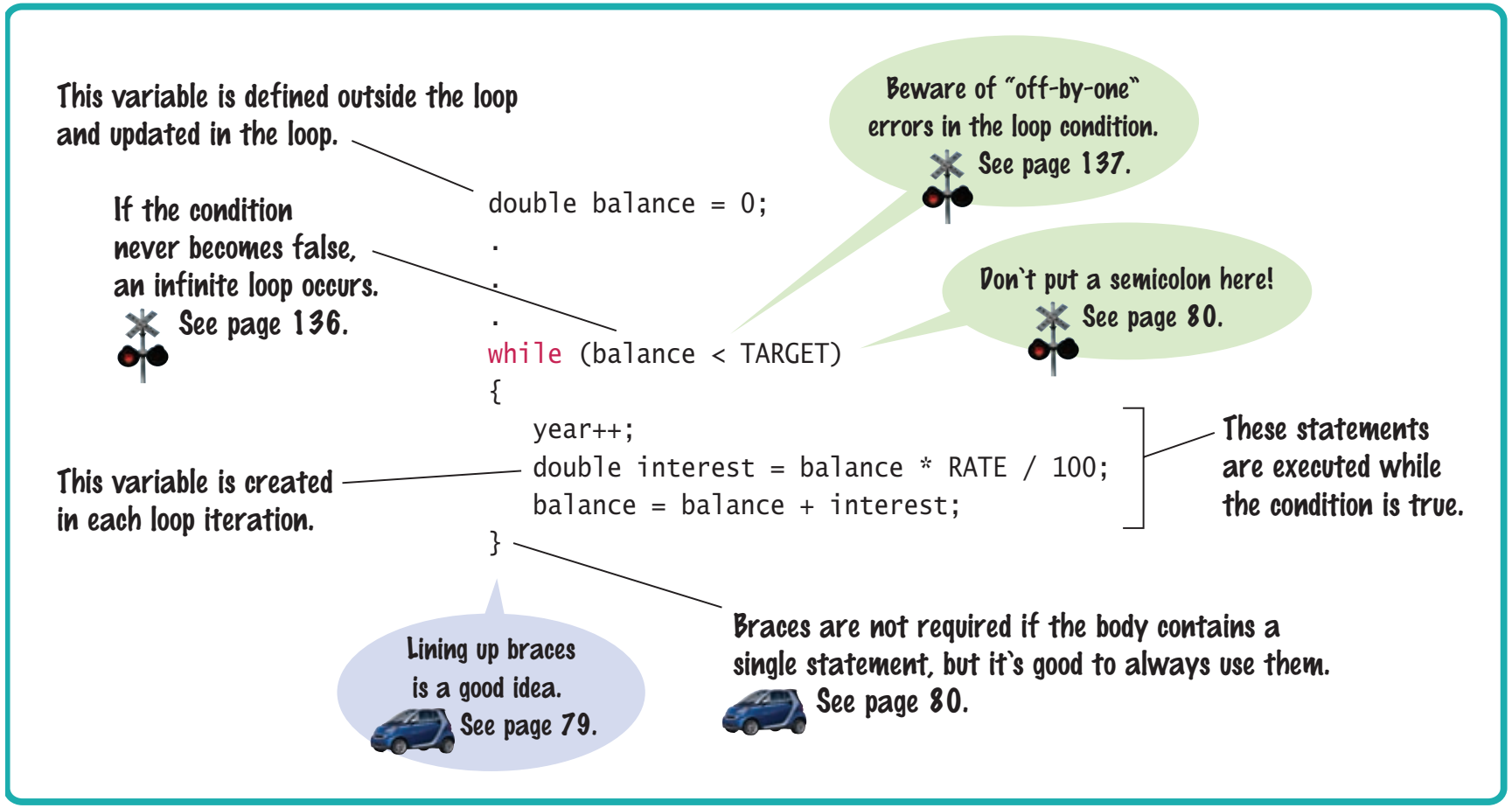
```
int digit; //switch can only test int and char types
…  //digit variable gets set here by some code
switch(digit)
{
        case 1: digit_name = "one"; break;
        case 2: digit_name = "two"; break;
        case 3: digit_name = "three"; break;
        case 4: digit_name = "four"; break;
        case 5: digit_name = "five"; break;
        case 6: digit_name = "six"; break;
        case 7: digit_name = "seven"; break;
        case 8: digit_name = "eight"; break;
        case 9: digit_name = "nine"; break;
        default: digit_name = ""; break; //taken if none of the above
}
```

`break` statements in the `switch` statement

- Every branch of the switch must be terminated by a break statement. And each branch must terminate with a semicolon.
- break tells the machine to skip down to the end of the switch statement, because a match was found.
- If the break is missing, execution falls through to the next branch, and so on, until finally a break or the end of the switch is reached.
- In practice, this fall-through behavior is rarely useful, and it is a common cause of errors.
- If you accidentally forget the break statement, your program compiles but executes unwanted code. Try it and see!

Loops

The *while* Loop Syntax



while Loop Examples		
Loop (all preceded by i=5;)	**Output**	**Explanation**
while (i > 0) { cout << i << " "; i--; }	5 4 3 2 1	When i is 0, the loop condition is false, and the loop ends.
while (i > 0) { cout << i << " "; i++; }	5 6 7 8 9 10 11 ...	The i++ statement is an error causing an "infinite loop" (see Common Error 4.1).
while (i > 5) { cout << i << " "; i--; }	(No output)	The statement i > 5 is false, and the loop is never executed.

while (i < 0)
{ cout << i << " ";
i--;
}
	(No output)	The programmer probably thought, "Stop when i is less than 0". However, the loop condition controls when the loop is executed, not when it ends (see Common Error 4.2).

while (i > 0);
{ cout << i << " ";
i--;
}
	(No output, program does not terminate)	Note the semicolon before the {. This loop has an empty body. It runs forever, checking whether i > 0 and doing nothing in the body.

Example of Normal Execution

while loop to hand-trace

```
i = 5;
while (i > 0)
{
   cout << i << " ";
   i--;
}
```

What is the output?

```
..
```

Example of a Problem – An Infinite Loop

The output never ends

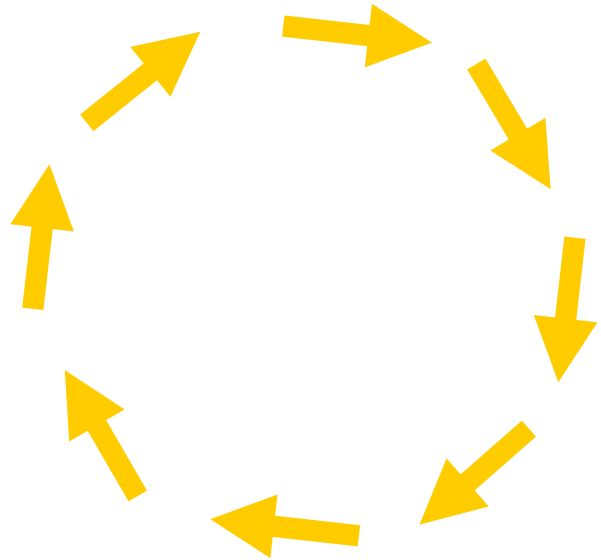
- *i* is set to 5
- The *i++;* statement makes *i* get bigger and bigger
- the condition will never become false –
- an infinite loop

```
i = 5;
while (i > 0)
{
   cout << i << " ";
   i++;
}
```

5 6 7 8 9 10 11…

Common Error – Infinite Loops

- Forgetting to update the variable used in the condition is common.
- In the investment program, it might look like this:



```
year = 1;
while (year <= 20)
{
   balance = balance * (1 + RATE / 100);
}
```

The variable **year** is not updated in the loop body!

Another Programmer Error

What is the output?

```
i = 5;
while (i < 0)
{
   cout << i << " ";
   i--;
}
```

A Very Difficult Error to Find (especially after looking for hours and hours!)

What is the output?

```
i = 5;
while (i < 0)
{
   cout << i << " ";
   i--;
}
```

do loop

The `do{ } while()` Loop

- The while() loop's condition test is the first thing that occurs in its execution.
- The do loop (or do-while loop) has its condition tested only after at least one execution of the statements. The test is at the bottom of the loop:

```
do
{
   statements
}
while (condition);
```

The do Loop

- This means that the do loop should be used only when the statements must be executed before there is any knowledge of the condition.

- This also means that the do loop is the least used loop.

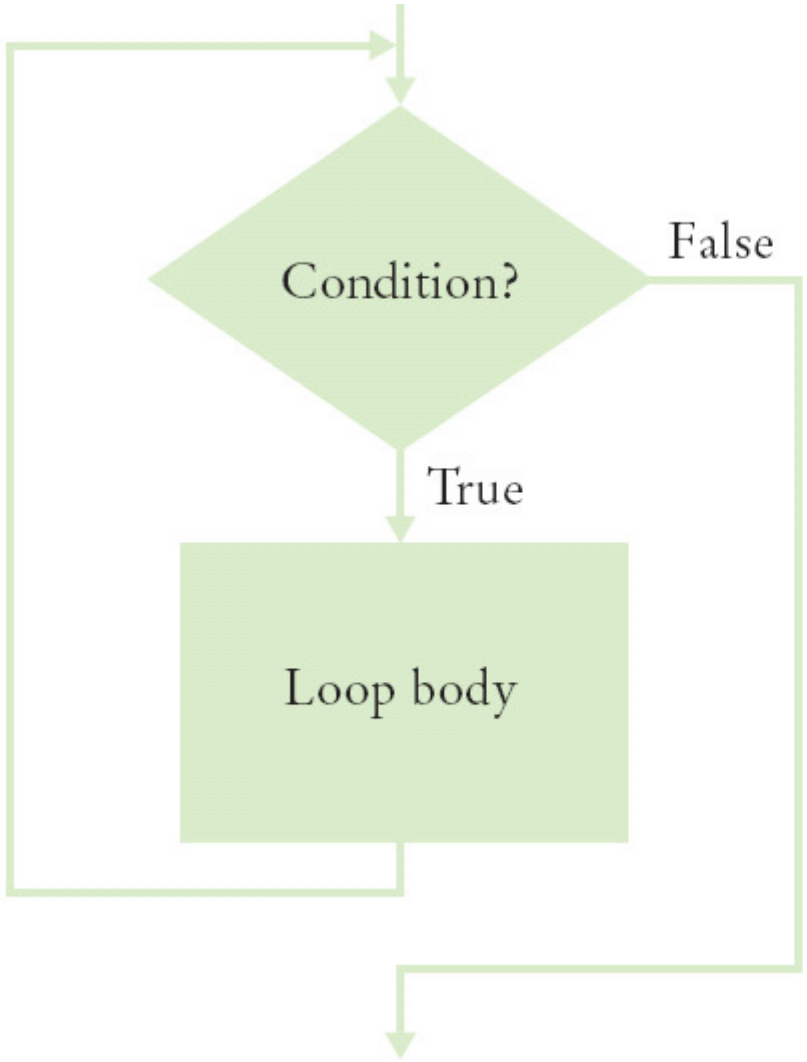
do { } Loop Code: getting user input Repeatedly

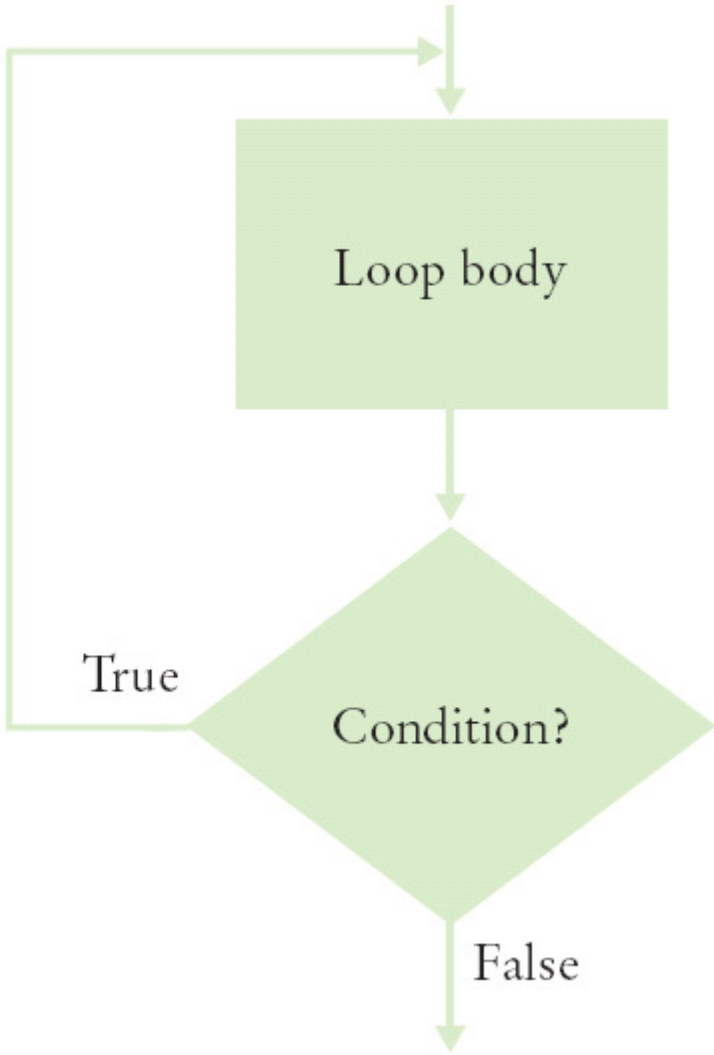
- Code to keep asking a user for input until it satisfies a condition, such as non-negative for applying the sqrt():

```
double value;
do
{
 cout << "Enter a number >= 0: ";
 cin >> value;
}
while (value < 0);

cout << "The square root is " << sqrt(value) << endl;
```

Flowcharts for the `while` Loop and the `do` Loop





Practice It: Example of `do...while`

- What output does this loop generate?

```
int j = 1;
do
{
   int value = j * 2;
   j++;
   cout << value << ", ";
} while (j <= 5);
```

How to Write a Loop

These are the steps to follow when turning a problem description into a code loop:

1. Decide what work must be done inside the loop
   - *For example, read another item or update a total*
2. Specify the loop condition
   - *Such as exhausting a count or invalid input*
3. Determine the loop type
   - *Use for in counting loops, while for event-controlled*
4. Set up variables for entering the loop for the first time
5. Process the result after the loop has finished
6. Trace the loop with typical examples
7. Implement the loop in C++