Loops

- A loop in an algorithm means that some of its steps are to be repeated several times
- There are many ways to express loops in algorithms and programs, we look at three different types that are commonly used:

  do-while-loops, while-loops and for-loops

- Loops are powerful, but they can also be the source of many programming errors when improperly used

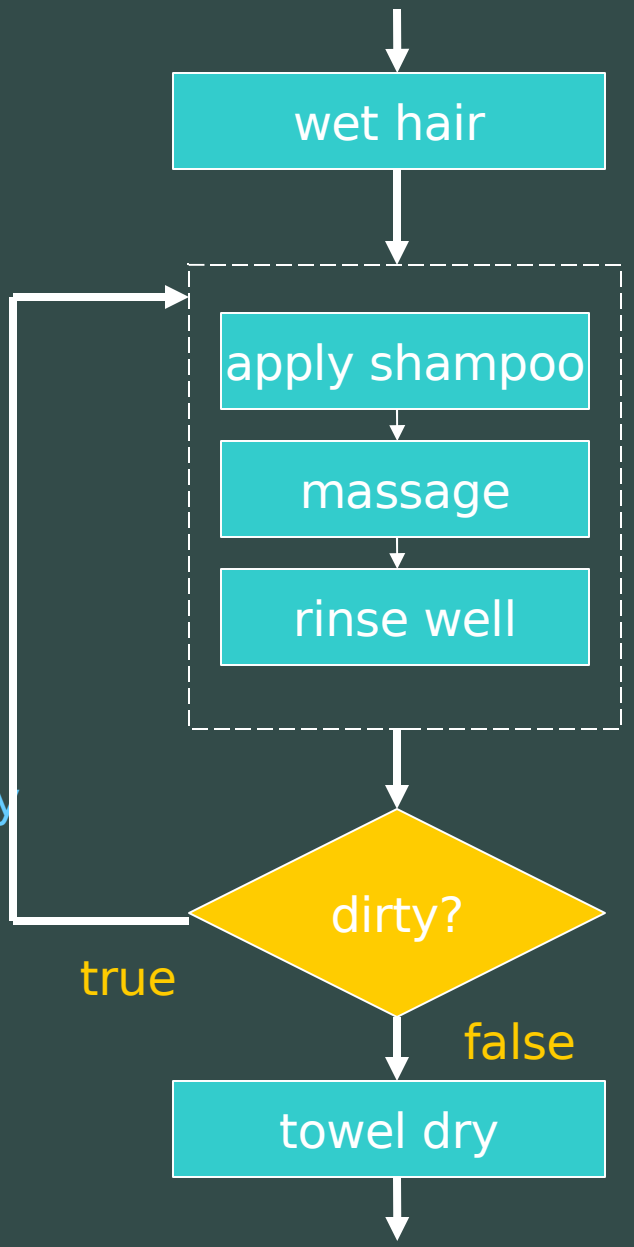
jmsamaniego@uplb.edu.ph (revised 2006)

An example of a **do-while** loop

```
{
  wet_hair;

  do {
     apply_shampoo;
     massage_into_hair_and_scalp;
     rinse_well;
  } while ( dirty );   // repeat if necessary

  towel_dry;
}
```



jmsamaniego@uplb.edu.ph (revised 2006)

do-while vs. **while** loops

- A do-while loop tests the condition after performing the steps, thus the steps in the body of the loop will be performed at least once
- A while loop tests before performing the loop body, thus they are executed zero or more times
- The difference is small, but it may be important in some applications

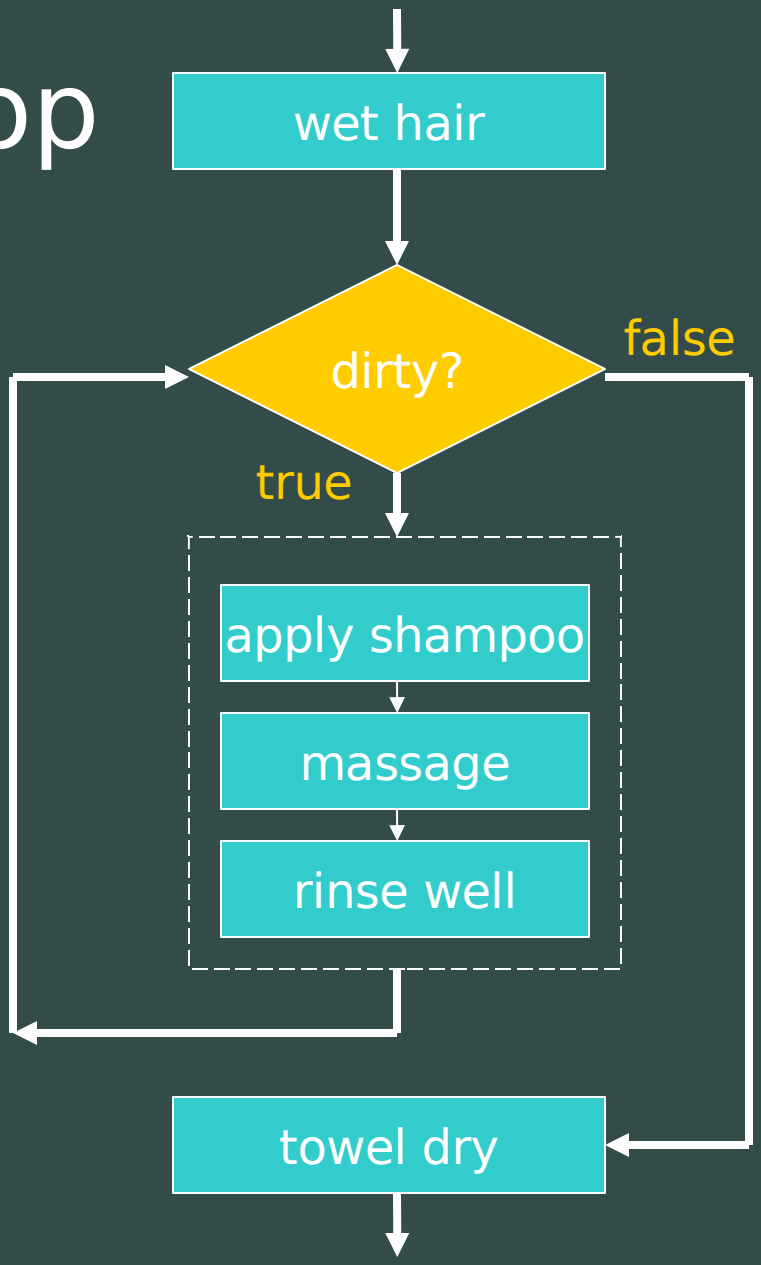
jmsamaniego@uplb.edu.ph (revised 2006)

Using a **while** loop

```
{
  wet_hair;

 while ( dirty )
  {
     apply_shampoo;
     massage_into_hair_and_scalp;
     rinse_well;
  }

 towel_dry;
}
```

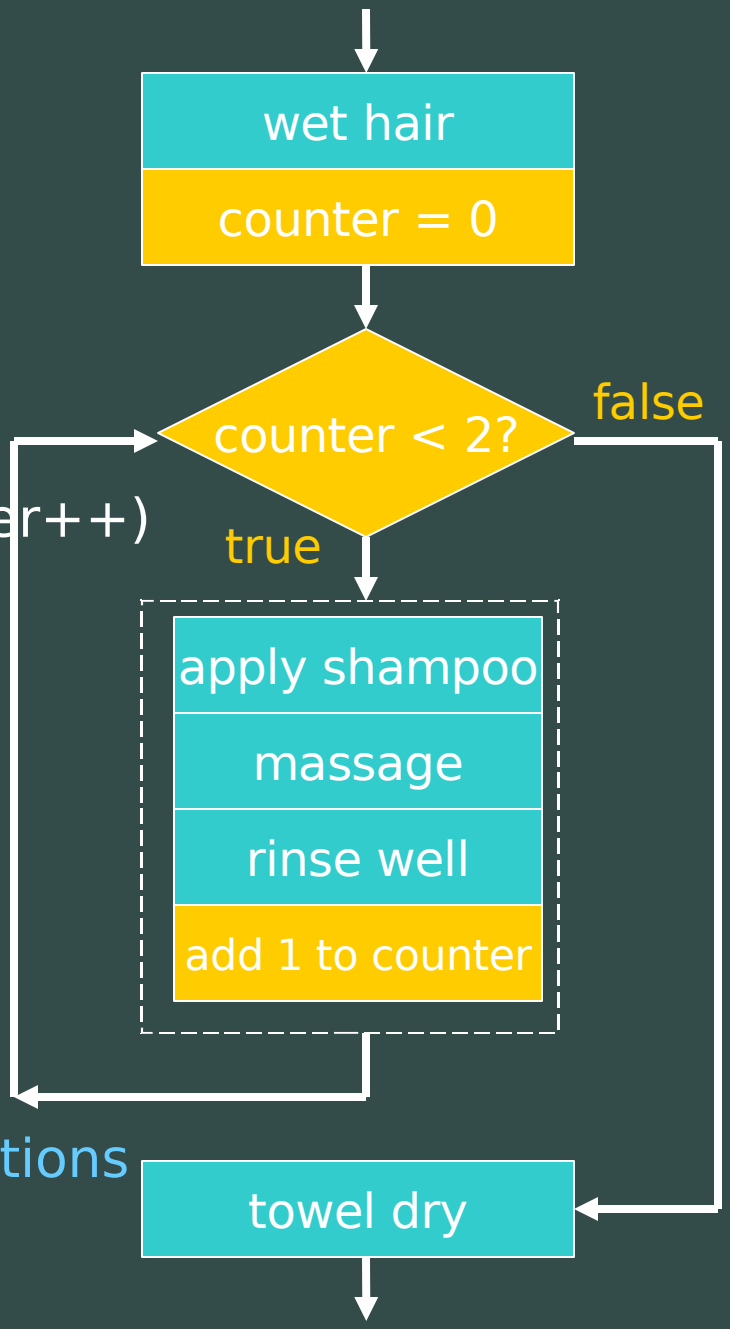


jmsamaniego@uplb.edu.ph (revised 2006)

for-loops are often used in bounded loops

```
{
  wet_hair;
  for (counter=0; counter<2; counter++)
  {
     apply_shampoo;
     massage_into_hair_and_scalp;
     rinse_well;
  }
  towel_dry;
}

// this particular example has 2 iterations
// (counter = 0, counter = 1)
```



jmsamaniego@uplb.edu.ph (revised 2006)

Sequence control keywords

- Branching
  - if, if-else, switch-case-break*
- Loops
  - do-while, while, for
  - break* and continue* inside loops

* to be discussed later and found in lab handouts

jmsamaniego@uplb.edu.ph (revised 2006)

Basic data types

- int
  - for whole integers (positive, negative or zero)
  - examples: 123, 0, -1
- char
  - for single characters (letters, digits, punctuation marks, special characters)
  - examples: ‘a’, ‘A’, ‘2’, ‘3’, ‘?’, ‘@’, ‘\n’

jmsamaniego@uplb.edu.ph (revised 2006)

Basic data types

- float
  - for numbers with decimal points
  - examples: 1.234 0.001 -65.4321
  - a special form of scientific notation is allowed for expressing large or small numbers
  - examples: 1.2e3 = 1200 -4.321e-2 = -0.04321
- double
  - similar to float, but allows more decimal places for higher precision

jmsamaniego@uplb.edu.ph (revised 2006)

Variables and assignment

- A variable is a named location in the computer's memory which can be assigned some value
- The current value can be changed any number of times during the execution of a program

```
#include <stdio.h>
main()
{
  int a = 0;
  printf("a contains %d\n", a); // prints: a contains 0
  a = 1;
  printf("a contains %d\n", a); // prints: a contains 1
}
```

a | 0

print as a decimal integer, followed by the newline character '\n'

jmsamaniego@uplb.edu.ph (revised 2006)

Basic input and output

- Values of variables can be read from the keyboard using the scanf() statement
  - for the basic data types, prefix the variable name with an ampersand '&'
- Values of variables or expressions can be printed to the screen using the printf() statement

```
#include <stdio.h>
main()
{
   int num;
   printf("Enter any integer: ");
   scanf("%d", &num);
   printf("The next one after that is %d\n",
   num+1);
}
```

jmsamaniego@uplb.edu.ph (revised 2006)

Integer operations and expressions

- Add +, subtract -, multiply *, divide /, remainder %

  ex. 5+3 = 8 8/2 = 4 7/2 = 3 8%2 = 0 8%3 = 2
- { multiplication, division, remainder } have higher precedence over { addition, subtraction }

  ex. 5+3*2 = 11 4+3/2 = 5 -7-5%2 = -7-1 = -8
- Unary minus (negation) has the highest precedence
- Evaluate left-to-right for operations within the same class

  ex. 4-3-2 = -1 8/2/2 = 2
- To override the normal rules of precedence, group expressions using pairs of parentheses

  ex. (5+3)*2 = 16 (4+3)/2 = 3 (7-5)%2 = 0
- Pairs of parentheses can be nested any number of levels

  ex. (4-(3-2))*(8/(2/2)) = 3*8 = 24

jmsamaniego@uplb.edu.ph (revised 2006)

The assignment statement

- A typical assignment statement has the form

  variable = expression;

- The expression is evaluated and the result is assigned as the new value of the variable

```
cm = 2.54 * inches;    /* convert to metric system */
C = (F - 32.0) * 5.0/9.0;  /* convert F to Celsius */
x = x + 1;             /* increase the value of x by 1 */
x = -x;                /* change the sign of x */
```

jmsamaniego@uplb.edu.ph (revised 2006)

Commonly used abbreviations

- An integer variable m can be incremented using m++ (or by ++m), and decremented using m-- (or by --m)

m++;	is equivalent to	m = m + 1;
m--;		m = m - 1;
m += 2;		m = m + 2;
m -= 3;		m = m - 3;
m *= 2;		m = m * 2;
m *= factor;		m = m * factor;

jmsamaniego@uplb.edu.ph (revised 2006)

Conditions

- Numbers (int, float, double) can generally be compared with similar types using the operators

  == equal != not equal

  < less than <= less than or equal

  < greater than >= greater than or equal

- Ex.
```
if (a==b) {
        printf("the 2 values are equal");
}
```
- Avoid using == to compare floats or doubles because of round-off errors, instead compute the absolute difference and test if it is small enough, if (abs(a-b) < 0.000001) ...

jmsamaniego@uplb.edu.ph (revised 2006)

Putting together sequence control, variables, expressions, and input/output

```
/* Input any integer, output its absolute value */
#include <stdio.h>
main()
{
  int num;
  printf("Enter any integer: ");
  scanf("%d", &num);
  if (num > 0) {
    printf("absolute value is %d\n", num);
  }
  else {
    printf("absolute value is %d\n", -num);
  }
}
```

jmsamaniego@uplb.edu.ph (revised 2006)

Compound conditions

- Use ! (not), && (and), || (or) to form more complex conditions when appropriate

```
if ((20 < temp) && (temp < 28)) {
    printf(”that temperature is nice”);
}
```

jmsamaniego@uplb.edu.ph (revised 2006)

Some practical guidelines

- Be sure you understand the problem very well.
- For non-trivial tasks, remember to start with the design of an algorithm on paper.
- Don’t rush to the keyboard without designing an algorithm first, and hand-checking it for a variety of valid inputs. Real programmers prove the correctness of their algorithm.
- Learn the intricacies of the programming language by reading well-written examples in books and practicing regularly.
- Everyone makes mistakes, but beginners naturally commit more errors than experienced programmers.
- Searching for and correcting errors in your algorithm or program (also known as “debugging”) is a skill that you also need to learn.

jmsamaniego@uplb.edu.ph (revised 2006)

Exercises (use if, if-else)

- Input the temperature in Celsius, and print if it is too cold for you, too hot for you, or just right (also print its equivalent in Fahrenheit for those people who still insist on using this silly non-metric system)
- Input any 3 positive integers A, B, C in any order and determine if the sum of any two of them equals the third
- Input any 3 positive integers A, B, C in any order, and determine the type of triangle that can be formed (if at all possible) with those values as sides (some possible triangle types include equilateral, isosceles, or right)

jmsamaniego@uplb.edu.ph (revised 2006)

Exercises
(try using different types of loops)

- Input 2 integers A and B, and output all the odd integers greater than A but less than or equal to B
- Input N, followed by N numbers using a loop and find their sum and average
- Input any positive integer N, and output the product 1*2*3*...*N (known as “N factorial” or N!)
- Input any 2 positive integers A and B and find their least common multiple
- Try Newton's algorithm to find the square root of x
- Revise Newton's algorithm to find the fourth root of x
- Revise Newton's algorithm to find the cube root of x

jmsamaniego@uplb.edu.ph (revised 2006)