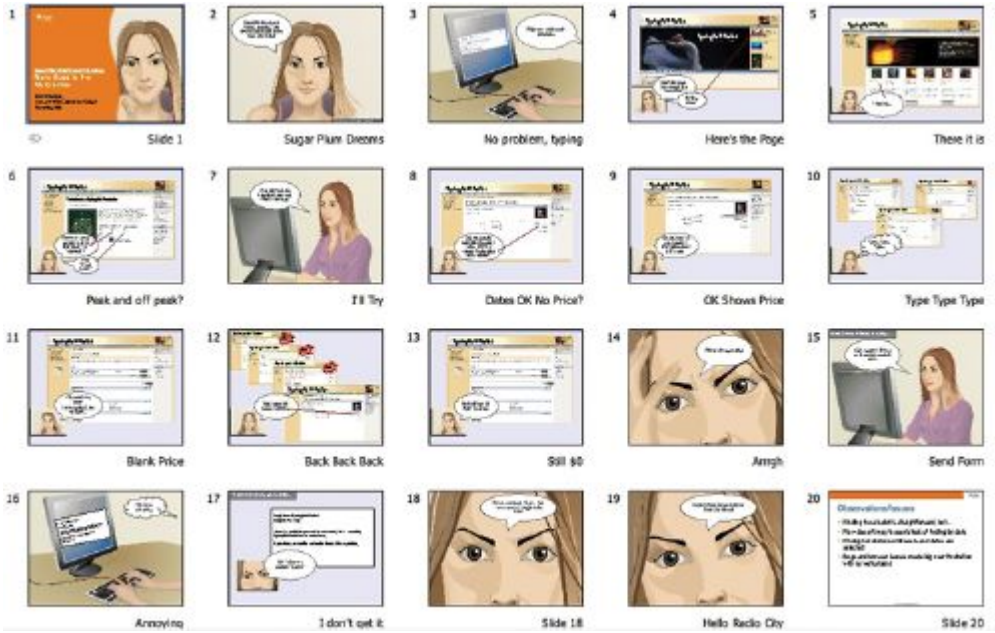
Topic 6

1. The `while` loop
2. Problem solving: hand-tracing
3. The `for` loop
4. The `do` loop
5. Processing input
6. Problem solving: storyboards
7. Common loop algorithms
8. Nested loops
9. Problem solving: solve a simpler problem first
10. Random numbers and simulations
11. Chapter summary

Problem Solving: Storyboards for User Interaction

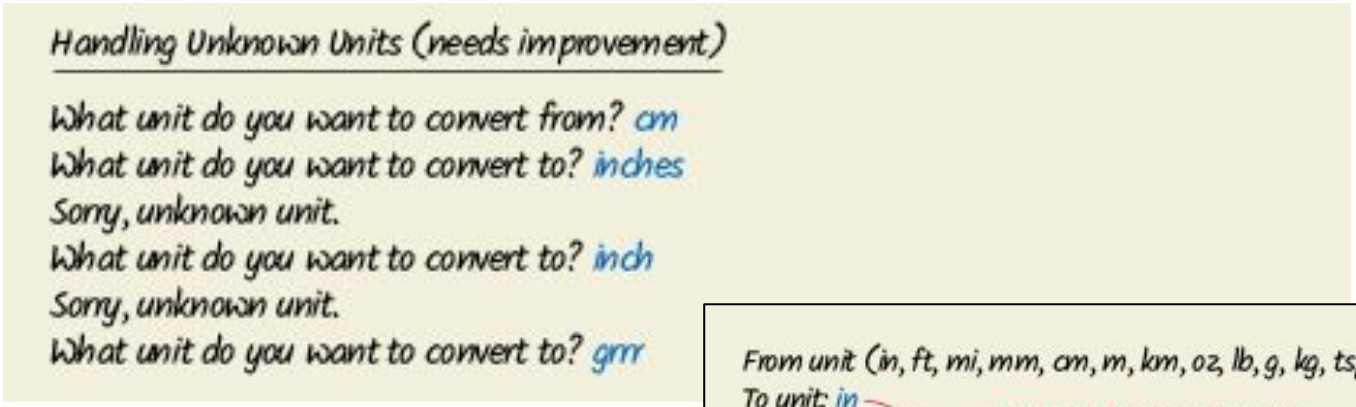
- To plan the user interface of your program, you can use a series of pictures or pseudocode showing the sequence of user output/input
  - This process forces you to think through possible scenarios
  - It leads to a better program that is less likely to require a re-write after user testing.

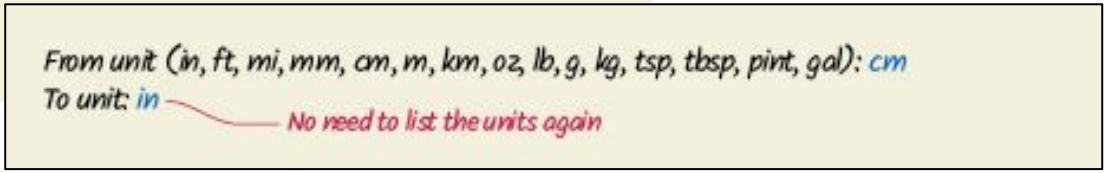
- Below is a storyboard for an app or game, but since our programs so far are only text, your storyboards will be words only



Storyboarding

- Ask yourself:
  - What inputs does the program need?
  - How will it ask the user for the inputs?
  - What outputs will the program display?
  - What is the best way to display the outputs?
- Below are example storyboards (before and after) for a program to convert values from one unit (such as inch) to another (such as cm).
  - Based on the storyboard, the programmer decided to display a list of possible units rather than assume the user knows them a priori.





Topic 7

1. The `while` loop
2. Problem solving: hand-tracing
3. The `for` loop
4. The `do` loop
5. Processing input
6. Problem solving: storyboards
7. Common loop algorithms
8. Nested loops
9. Problem solving: solve a simpler problem first
10. Random numbers and simulations
11. Chapter summary

Common Loop Algorithms

- These examples provide a starting point for your code

- Total and average of user inputs:

```
double total = 0;
int count = 0;
double input;
while (cin >> input)
{
   total = total + input; //compute running total
   count++;
}
double average = 0;
if (count > 0)
{
   average = total / count;
}
```

Common Loop Algorithms: Counting Matches

```
//Counting chars in a string
int spaces = 0;
for (int i = 0; i < str.length(); i++)
{
   if (str.substr(i, 1)== " ")
   { spaces++; }
}

//Counting words in a user input sequence
int short_words = 0;
string input;
while (cin >> input)
{
   if (input.length() <= 3)
   { short_words++; }
}
```

Common Loop Algorithms: Finding First Location

```
//Find the location in a string of first space char
bool found = false; //flag=false says "not found yet"
int position = 0;
while (!found && position < str.length())
{
   string ch = str.substr(position, 1);
   if (ch == " ")
      { found = true; }
   else
      { position++; }
}
```

Common Loop Algorithms: Prompting Until Matched

```
//Repeat prompt until user enters valid value

bool valid = false; //input not valid yet
double input;  //declare input var outside loop,
         //so it will persist after loop exit
while (!valid)
{
   cout << "Please enter a positive value < 100: ";
   cin >> input;
   if (0 < input && input < 100)
      { valid = true; }
   else
      { cout << "Invalid input." << endl; }
}
```

Common Loop Algorithms: Min and Max

```
//Save the min and max values of user input list
// This is a merger of the min and max loops from book

double largest, smallest;
double input;
cin >> largest; //get first value to use in loop
smallest = largest; // copy it.
// If only 1 entry, it is both smallest and the largest

while (cin >> input)
{
   if (input > largest)
   { largest = input; }
   else if (input < smallest)
   { smallest = input; }
}
```

Common Loop Algorithms: Comparing Adjacent Values

```
//Find adjacent duplicates of user input list
// In a later chapter, we'll show how to use arrays to
// find non-adjacent duplicates

double input;
double previous; //to keep track of prior entry
cin >> previous; //first entry becomes first previous
while (cin >> input)
{
   if (input == previous)
   {
      cout << "Duplicate input" << endl;
   }
   previous = input; //save it to compare to next input
}
```

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++

Worked Example 4.1: Loop to Remove Chars from `string`

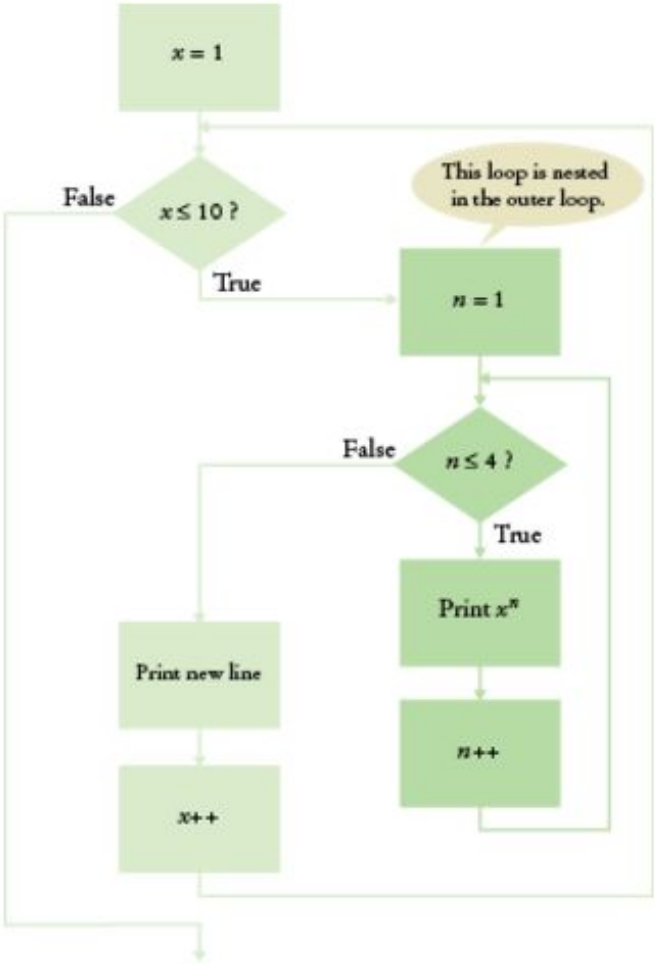
```
// worked_example_1/ccnumber.cpp
// Removes all spaces or dashes from a string
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string credit_card_number = "4123-5678-9012-3450";
    int i = 0;
    while (i < credit_card_number.length())
    {   string ch = credit_card_number.substr(i, 1);
        if (ch == " " || ch == "-") //must remove char
        {
            string before = credit_card_number.substr(0, i);
            string after = credit_card_number.substr(i + 1);
        credit_card_number = before + after;
        }
        else // no need to remove it, go to next char
        { i++; }
    }
    cout << credit_card_number << endl;
    return 0;
}
```

Topic 8

1. The `while` loop
2. Problem solving: hand-tracing
3. The `for` loop
4. The `do` loop
5. Processing input
6. Problem solving: storyboards
7. Common loop algorithms
8. Nested loops
9. Problem solving: solve a simpler problem first
10. Random numbers and simulations
11. Chapter summary

Nested Loops

- Nested loops are used mostly for data in tables as rows and columns
- The processing across the columns is a loop, as you have seen before, "nested" inside a loop for going down the rows.
  - Each row is processed similarly. After writing a loop to process a generalized row (across the columns), that loop, called the "inner loop," is placed inside an "outer loop" that does successive rows
- The flowchart shows a diamond decision box for each loop.



Nested Loop Example: Table of Powers

Write a program to produce a table of powers.
The output should be something like this:

1	2	3	4
X	X	X	X
1	1	1	1
2	4	8	16
3	9	27	81
4	16	64	256
5	25	125	625
6	36	216	1296
7	49	343	2401
8	64	512	4096
9	81	729	6561
10	100	1000	10000

Nested Loop Inner Loop

- There are four columns and in each column we display the power. Using x to be the number of the row we are processing, we have (in pseudo-code) for the "inner" loop:

```
For n from 1 to 4  //table row
    Print x^n
```

You should test that this works in your code before continuing. If you can't correctly print one row, why try printing lots of them?

Nested Loop Outer Loop

We add the outer loop to count the rows, and include the inner loop to print each row:

```
Print table header.
For x from 1 to 10
   For n from 1 to 4  //table row
      Print x^n
   Print endl.
```

Nested Loop Program for Table of Powers

ch04/powtable.cpp

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

void main() //print a table of powers, x to the nth
{
   const int NMAX = 4;
   const double XMAX = 10;
   for (int n = 1; n <= NMAX; n++) // Print table header
      cout << setw(10) << n;
   cout << endl;
   for (int n = 1; n <= NMAX; n++)
      cout << setw(10) << "x ";
   cout << endl << endl;
   for (double x = 1; x <= XMAX; x++) // Print table row
   {
      for (int n = 1; n <= NMAX; n++) //print each column
         cout << setw(10) << pow(x, n);
      cout << endl;
   }
}
```

More Nested Loop Examples

The loop variables can have a value relationship. In this example the inner loop depends on the value of the outer loop.

```
for (i = 1; i <= 4; i++)
    for (j = 1; j <= i; j++)
        cout << "*";
    cout << endl;
```

The output will be:

```
*
**
***
****
```

Nested Loop Example: Triangle

```
for (i = 1; i <= 4; i++)
   for (j = 1; j <= i; j++)
      cout << "*";
   cout << endl;
```

`j` is *each* line's length, which is different for each line. and depends on the current line number, `i`. **Having `j` count up to `i` in the inner loop results in a longer line at each row.**

```
Output:
*
* *
* * *
* * * *
```

Nested Loop: Array of Numbers (Practice It #3)

What does the following code print?

```
for (int i = 1; i <= 4; i++)
{
   for (int j = 0; j < 4; j++)
   {
      cout << " " << 10 * i + j;
   }
   cout << endl;
}
```

The answer is below, in small font. Enlarge it to check your answer:

```
10 11 12 13
20 21 22 23
30 31 32 33
40 41 42 43
```

Worked Example 4.2: Pixels in an Image

- A digital image is made up of pixels.
  - Each pixel is a tiny square of one color
  - They are arranged in a 2D array of rows and columns
  - A pixel color is 3 bytes: one each for Red, Green, Blue intensity
- We will use a `Picture` type library (a `class`, to be covered in a later chapter)
  - It has functions for loading an image and accessing its pixels.

Problem Statement:

- Convert an image into its negative, turning white (255,255,255) to black (0,0,0), cyan to red, and so on.
  - To do so, we subtract the color values from 255 (the max possible color value)
  - We need nested loops to process the 2D array of pixels

Worked Example 4.2: Code

```
#include "picture.h"
int main()
{
   Picture pic("queen-mary.png"); //load the image
   for (int x = 0; x < pic.width(); x++) //outer, row loop
   {
      for (int y = 0; y < pic.height(); y++)//inner column
loop
      {
         int red = pic.red(x, y); //get individual RGB
values
         int green = pic.green(x, y);
         int blue = pic.blue(x, y);
         pic.set(x, y, 255 - red, 255 - green, 255 - blue);
      }
   }
   pic.save("out.png");
   return 0;
}
```