COMSC-110 Chapter 7

Do overs: Repetition or Looping

Valerie Colber, MBA, PMP, SCPM

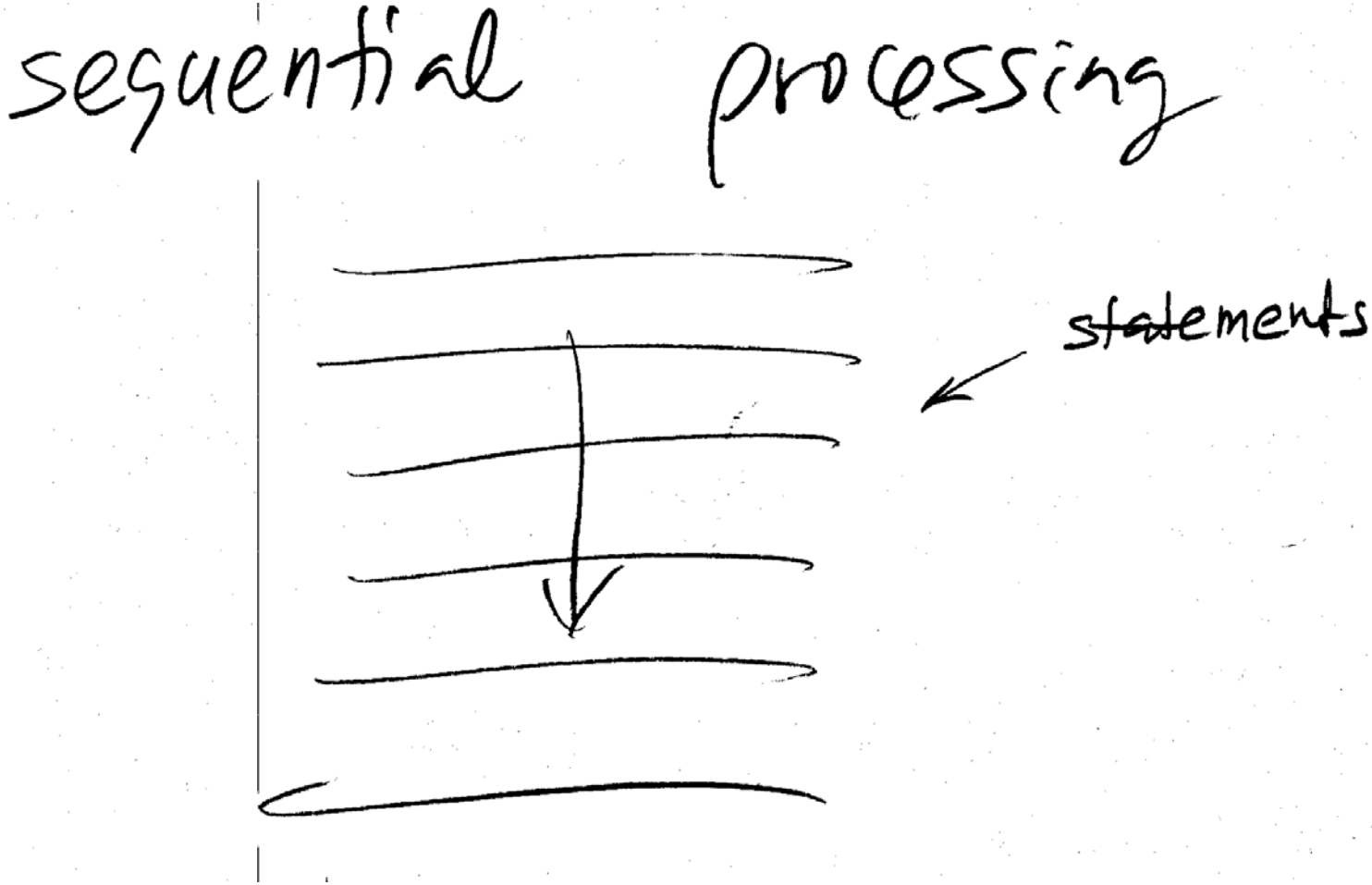
REVIEW
FROM
CHAPTER 6

3 control structures

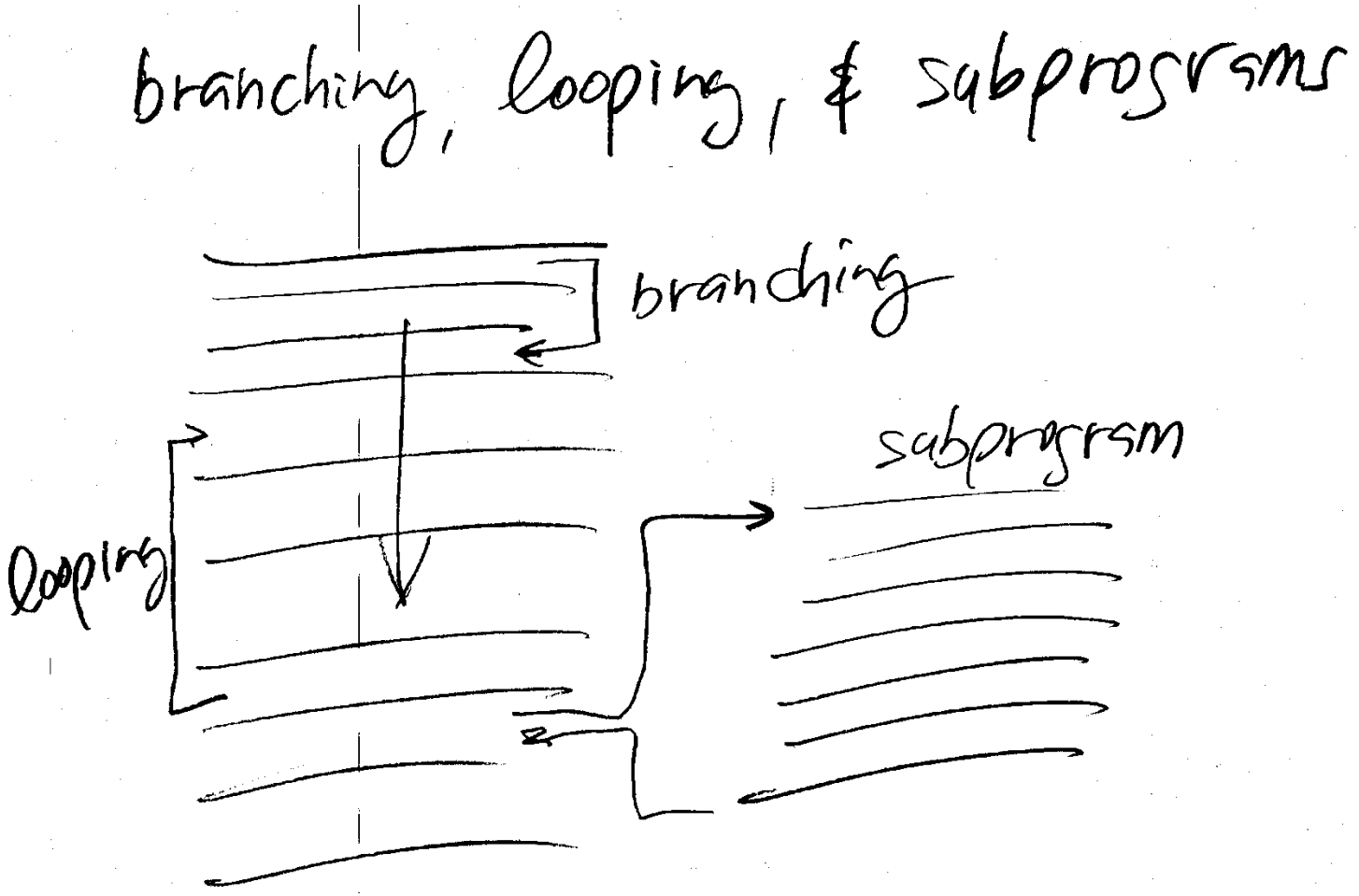
1. Sequence
2. Selection
3. Looping or Repetition

Flow of control

Sequential processing



Branching, looping, & subprograms



Exceptions to sequential processing

- Using logic, code blocks could be skipped over, using "if" and "switch" statements
- Using a loop, jump back up to a statement that was already processed

Kinds of loops

- Event controlled
- Count controlled

Syntax: The while statement

```
while (true)
{
the 1st statement in the loop
another statement in the loop
...
another statement in the loop
the last statement in the loop
}
```

Go back and re-entry point

- the "go back" syntax: a closing curly brace (})
- the "reentry point" syntax: two lines:

```
while (true)
{
```

- marking with // comments

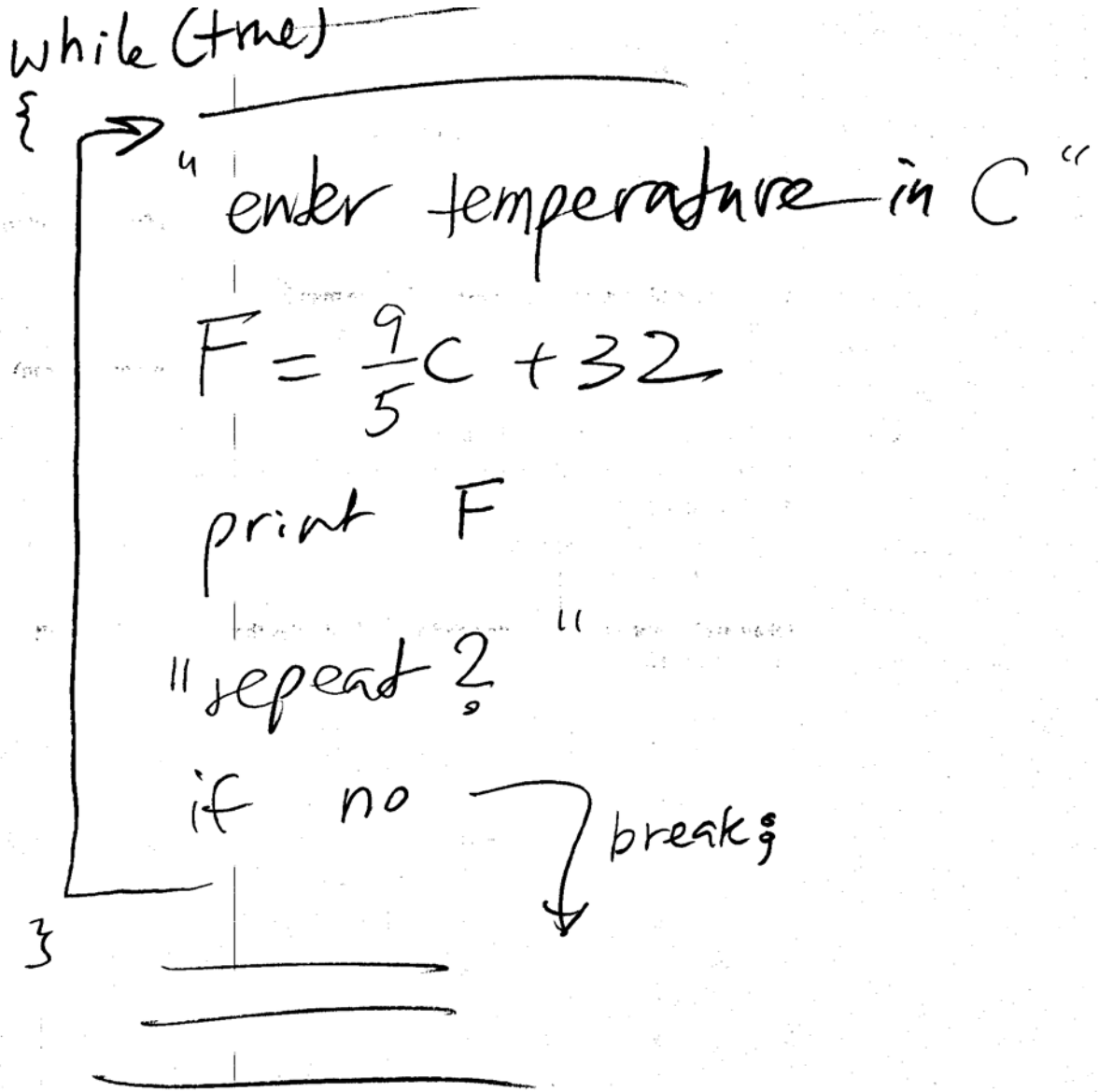
```
} // while
```

Scope

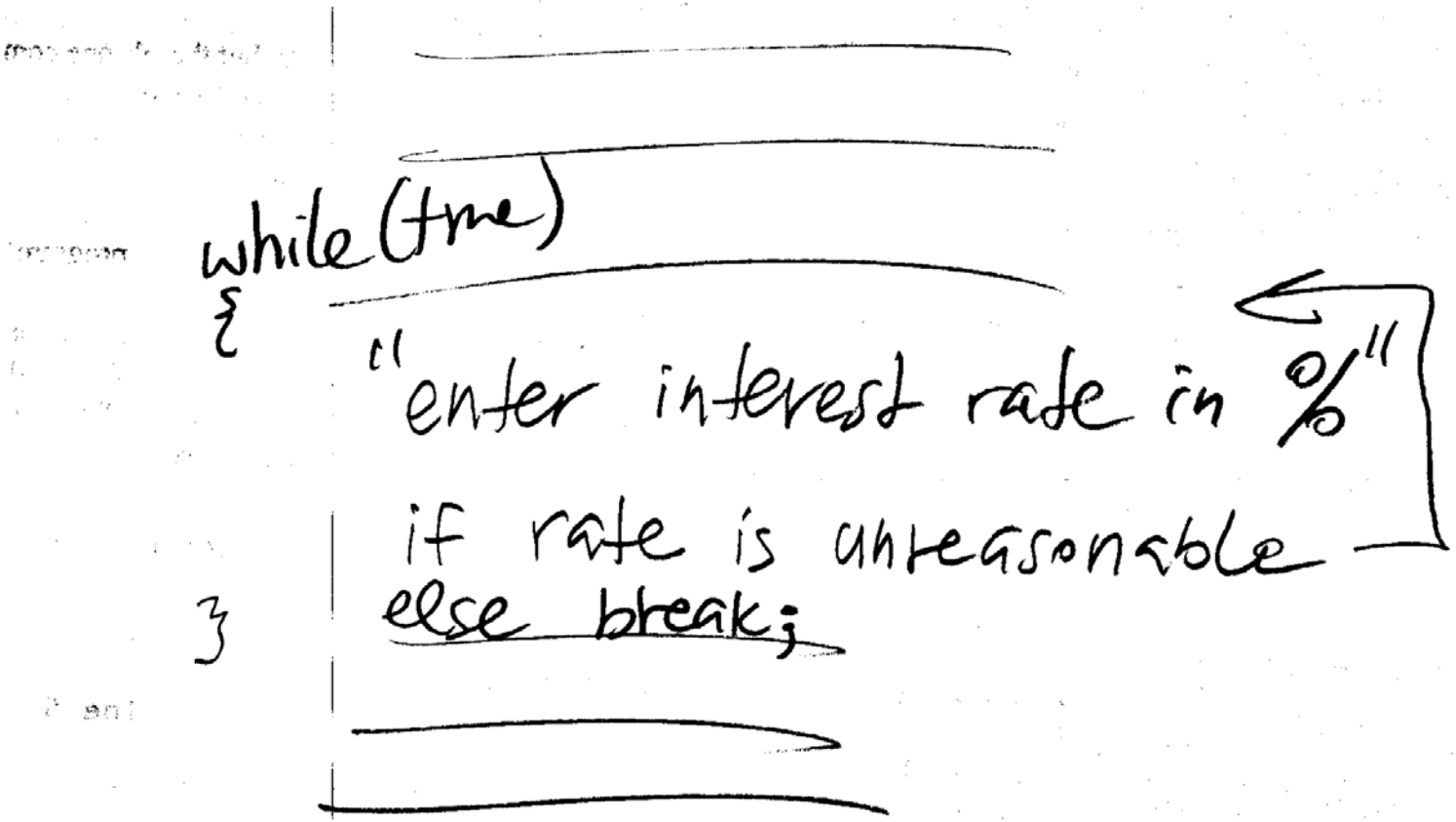
- A variable declared inside a container (curly braces or code block) is valid only in the container.

```
{
   ...
   int a;
   "a" is useable here...
}
...but not here
```

Repeat temperature calculation



Unreasonable interest rate



passingGradeL.cpp

```
#include <iostream>
using namespace std;
int main()
{
  while (true)
  {
    char grade; // read a char value from the keyboard
    cout << "What is your grade? [A, B, C, D, or F]: ";
    cin >> grade;
    cin.ignore(1000, 10);
    if (grade == 'A' || grade = 'B' || grade == 'C')
    {
      cout << "You pass" << endl;
    }//if grade is an A or B or C
  }//while true
}//main
```

What is wrong with passingGradeL.cpp?

- This loops over and over, actually never ending!
- Variable declared inside loop.
- Missing == in one of the conditions
- The only way out of this loop is to press CTRL-C instead of typing a grade when prompted! There really should be user instructions that tells the user to "Use CTRL-C to quit...". But relying on CTRL-C to end a program is really not good programming practice.

Infinite loop

- An infinite loop repeats over and over, to infinity.
- There is no way to end the program normally.
- CTRL-C ends it, but it ends the program, not just the loop.
- Some textbooks and instructors teach that you should never write an infinite loop.
- Actually, there is nothing wrong with having an infinite loop in your program -- you just don't want to get *stuck* in an infinite loop.
- That means you need to leave yourself a way out of the loop, and that's where the **if** - **break** statement comes in.

If-break statement

- The purpose of "if-break" is to define a condition under which looping would discontinue, and sequential processing would pick up at the first statement after the loop.

```
while (true)
{
  ...
  if (...) break;
  ...
}//while
```

passingGradeIB

```
…
#include <iostream>
using namespace std;
…
int main()
{
  //data
  char grade; //represents student's grade

  //check grade
  while (true)
  {
     cout << "What is your grade? [A, B, C, D, F, or X to quit]: ";
     cin >> grade;
     cin.ignore(1000, 10);
     if (grade == 'X' || grade == 'x') break;
     if (grade == 'A' || grade == 'B' || grade == 'C')
     cout << "You pass" << endl;
  } // while
  cout << "Thanks for checking your grades!" << endl;

} // main
```

What is X called?

Sentinel Value

- Note the minor enhancement added to this program -- it accepts uppercase *or* lowercase X to end the loop.
- This is what's known as a *sentinel* -- it signifies the end of a process.
- The program also prints a thank you note before the program ends.
- example: "Enter another number, or -1 to exit"
  if (n == -1) break;

Curly Braces and Comments

- If you are good about always aligning each closing curly with its matching opening curly brace, and indenting code blocks inside curly braces, then it will always be possible to find the match to a curly brace by scrolling through your code.
- But some programmers like to label their closing curly braces.
- Always label you ending curly brace with a comment as to what the code block it is for.

Exiting the loop

- syntax: the break; statement
- inside the {code block}, you need this:

```
if (...)
  break;
```

- ... stands for a condition:
- when it's met, quit the loop
  sequential control resumes AFTER closing curly brace

Event vs. Count controlled loops

And applying opposite and not logic

Event Controlled Loops

```
string answer;
while (true)
{
  cout << "Your answer [yes/no]: ";
  cin >> answer;
  if (answer == "yes")
    break;
  if (answer == "no")
    break;
  cout << "Let's try this again.\n";
}
... // resume here after "break"
```

if-break VS switch-break

```
…
#include <iostream>
using namespace std;
…
int main()
{
  //data
  char answer;

  //say it again?
  while (true)
  {
    cout << "Hello" << endl;

    cout << "Again? [Y/N]: ";
    cin >> answer;
    cin.ignore(1000, 10);

    switch (answer) // BAD IDEA BECAUSE OF break;
    {
      case 'N': case 'n':
        break; // WRONG CONTEXT -- BREAKS FROM SWITCH, NOT FROM THE LOOP!
    }//switch answer
  }//while true
  cout << "Finished!" << endl;

}//main
```

Validation loop

- Validation loops are used with console input, to make sure that valid entries are made.
- These loops repeat until a valid value is received, and usually print a message after detecting invalid input and before doing another cycle of the loop.

Without validation

```
...
  //data
  char grade; //user's grade

  //check grade
  cout << "What is your grade? [A, B, C, D, F, or X to quit]: ";
  cin >> grade;
  cin.ignore(1000, 10);
...
```

With validation

```
…
//data
char grade;

//check grade with input validation
while (true)
{
   cout << "What is your grade? [A, B, C, D, F, or X to quit]: ";
   cin >> grade;
   cin.ignore(1000, 10);
   if (grade == 'A' || grade == 'B' || grade == 'C' || grade == 'D' || grade == 'F' || toupper(grade) == 'X') break;
   cout << grade << " is an invalid grade. Try again..." << endl;
} // while
```

Reverse logic

- Logical expressions evaluate to true or false. Sometimes when formulating a logical expression for a programming solution, the logic ends up being the opposite of what was intended.
- For example, you might come up with "if this condition happens, skip the next code block" when what you really wanted is "if this condition happens, *execute* the next code block".
- So you would have to reverse the "if" logic.
- Or you may find it necessary to rewrite a code block in such a way that the originally written logical expression needs to be reformulated as its opposite.
- For example, the original code block may say "password accepted", but want to change it to say "invalid password -- try again".

Using opposite operators

- If the original expression is a simple comparison with a logical operator, swap out the operator with its opposite. Swap **==** with **!=** , **<** with **>=** , or **>** with **<=** , and vice versa.

Using “not” logic

- No matter what the original expression is, you can always enclose it in parentheses and put a "not operator" in front.
- The "not operator" is an exclamation sign.
- So the logical opposite of **grade == 'A'** is **!(grade == 'A')** .
- The logical opposite of **grade == 'A' || grade == 'B' || grade == 'C'** is **!(grade == 'A' || grade == 'B' || grade == 'C')** .
- The logical opposite of **fin.good()** is **!(fin.good())** , which can actually be written as **!fin.good()** without parentheses because there is no logical operator in the expression.

Using DeMorgan's Theorem

- DeMorgan's Theorem can be applied to expressions that have "and" or "or" logic. It is done by swapping **==** with **!=** , **<** with **>=** , or **>** with **<=** (and vise versa), where they appear. Then swap **&&** with **||** (and vise versa).
- For example, the reverse of **answer == 'y' || answer == 'Y'** is **answer != 'y' && answer != 'Y'** .

Count Controlled Loops

- declare an int to track #of cycles
- break from loop after certain #of cycles

```
int i; // a cycle counter
i = 0; // initialize counter: zero cycles so far
while (true) // repeat...
{            // ...from here
  if (i == 10) // after 10th cycle...
    break; // quit
  ... // do something...
  i = i + 1; // that's one more cycle
} // go back to repeat cycle
... // resume here after "break"
```

3 ways to implement the repetition control structure in C++

Different ways to do loops in C++

1. While (condition)

- The condition is tested at the beginning of the loop so if the condition is false when the loop begins, it is possible the loop is not executed not even once.

2. Do --- While (condition)

- The condition is tested at the end of the loop so if the condition is false when the loop begins, the loop will always execute at least once.

3. For (set index, condition, increment/decrement)

- The loop executes for a set number of times

Choose the correct loop

- If you know how many times to loop, use the FOR loop.
- If you need the loop to execute at least once, even if the condition to stop looping is true, use the DO-WHILE loop.
- If you need the loop to not execute if the condition is false from the get go, use the WHILE loop
- Do NOT use the WHILE hard coded true condition with a break statement except in the rare circumstance it is required.

Shorthand notation

- i++ means i = i + 1
- i-- means i = i - 1

Bounce.cpp

```
//objective: simulate a bouncing ball from a height in inches
//libraries
#include <iostream>
using namespace std;
…
//main program
int main()
{
  //data
  double height = 40;  //initial ball height in inches
  int i=0; //ball bouncing loop index

  //ball bouncing simulation loop
  while (i < 10)
  {
     height = height / 2.0;
     cout << "location is now: " << height << " inches" << endl;
     i = i + 1;
  } // while
} // main
```

thereYet2.cpp

```
//objective: simulate kids in a car asking if we have arrived at our
destination yet
//libraries
#include <iostream>
#include <string>
using namespace std;
…
//main program
int main()
{
  //data
  string answer;  //user response

  //input validation loop
  do
  {
     cout << "Are we there yet? ";
     cin >> answer;
     cin.ignore(1000, 10);
  } while (answer != "yes");
} // main
```

Bounce2.cpp

```
//objective: simulate a bouncing ball from a height in inches
//libraries
#include <iostream>
using namespace std;
...
//main program
int main()
{
  //data
  double height = 40; //height of ball bounce in inches
  int i; //ball bouncing loop index

  //simulate ball bouncing from initial height
  for (i = 0; i < 10; i++)
  {
     height = height / 2;
      cout << "location is now: "       << height << " inches" << endl;
  } // for
} // main
```

Find min and max score

```
//objective: find the max AND min scores
….
//data
int sentinel = -999; // an unexpected score value
int max = sentinel; // signifies that max is not yet set
int min = sentinel; // signifies that min is not yet set
int aScore; //each user input score between 0 and 100


//find maximum and minimum values
while (true)
{
  cout << "Enter a score [between 0-100 or " << sentinel << " to exit]: ";
  cin >> aScore;
  cin.ignore(1000, 10);
  if (aScore == sentinel) break; // that's the signal to exit this loop
  if (max < aScore) max = aScore;
  if (min > aScore) min = aScore;
}//while true
…
```

partialAlphabet.cpp

```
//objective: output the English alphabet from letter A to user selected number of letters

//libraries
#include <iostream>
using namespace std;
…
//main program
int main()
{
  //data
  int n; // print this many letters
  char letter = 'A'; //first letter of the alphabet to begin output from
  int i; //output loop index

  //input user selected number of letters to output
  cout << "How many letters to print? ";
  cin >> n;
  cin.ignore(1000, 10);

  //output n number of letters of the alphabet starting at A
  for (i = 0; i < n; i++)
  {
    cout << letter;
    letter++; // becomes next letter
  } // for
  cout << endl;
} // main
```

backwards.cpp

```
//objective: output letters of English alphabet backwards from letter Z
…
//libraries
#include <iostream>
using namespace std;
…
//main program
int main()
{
  //data
  char letter = 'Z'; //letter to begin output of the alphabet backwards
  int i; //output loop index

  //output loop
  for (i = 0; i < 26; i++)
  {
    cout << letter;
    letter--; // becomes previous letter
  } // for
  cout << endl;
} // main
```

Nested Loops:
loops inside of loops

nestedforloop.cpp

```
#include <iomanip>
#include <iostream>
using namespace std;
int main()
{
  int times; // number of times to loop
  int row; // outer loop index
  int column; // inner loop index
  cout << "How many times to loop?";
  cin >> times;
  cin.ignore (1000, 10);

  for (row = 1; row <= times; row++)
  {
    for (column = 1; column <= times; column++)
    {
       cout << "row=" << row << ' ' << "column="
<< column << endl;
    } //for column inner loop
  } // for row outer loop
} // main
```

MEMORY

row	column	times
~~1~~	~~1~~	3
~~1~~	~~2~~	
~~1~~	~~3~~	
~~1~~	~~4~~	
~~2~~	~~1~~	
~~2~~	~~2~~	
~~2~~	~~3~~	
~~2~~	~~4~~	
~~3~~	~~1~~	
~~3~~	~~2~~	
~~3~~	~~3~~	
~~3~~	~~4~~	
4		

OUTPUT

```
C:\>nestedforloop
How many times to loop?3
row=1 column=1
row=1 column=2
row=1 column=3
row=2 column=1
row=2 column=2
row=2 column=3
row=3 column=1
row=3 column=2
row=3 column=3
```

See
digitalClock.cpp
program example

See
rightTriangle.cpp
program example

Pausing for one second

```
//Special compiler dependent definitions
#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif

…

//pause for one second
#ifdef _WIN32
Sleep(1000); // one thousand milliseconds
#else
sleep(1); // one second
#endif
```

1. Special compiler dependent definitions means special libraries dependent on operating system
2. The sleep function was not written for the same units of measure, which is why we need the right library for that operating system, so then to use the sleep function we have to do it based on the correct library dependent

How to write an algorithm with looping logic

- Many samples of algorithms, which are basically step-by-step instructions.
- To create a C++ program, you first write an algorithm in English, and then translate it to C++.
- Then you "compile" what you write in C++ in order to make a file containing a second translation of your algorithm that the computer can understand.

All loops look the same in an algorithm

- The only difference is whether the if instruction is at the beginning of the loop or at the end of the loop to test the condition for ending a loop.

4 forms of loop statement

1. Loop With Curly Brace Code Block
2. Loop Without Curly Brace Code Block
3. Loop All On One Line
4. Beware! Loop With Semicolon

Loop With Curly Brace Code Block

```
while (true)
{
...
... one or more
... statements
...
}
```

Single statement loop

while (...)
statement;

Loop all on one line

while (...) *statement*;

A Loop Statement That DOES NOT WORK AS EXPECTED!!!

```
while (...);
```

```
for (int i = 0; i < 10; i++);
  cout << "Hello!" << endl;
```

References

- INTRODUCTION TO PROGRAMMING USING C++ by Robert Burns