COMSC-110 Chapter 6

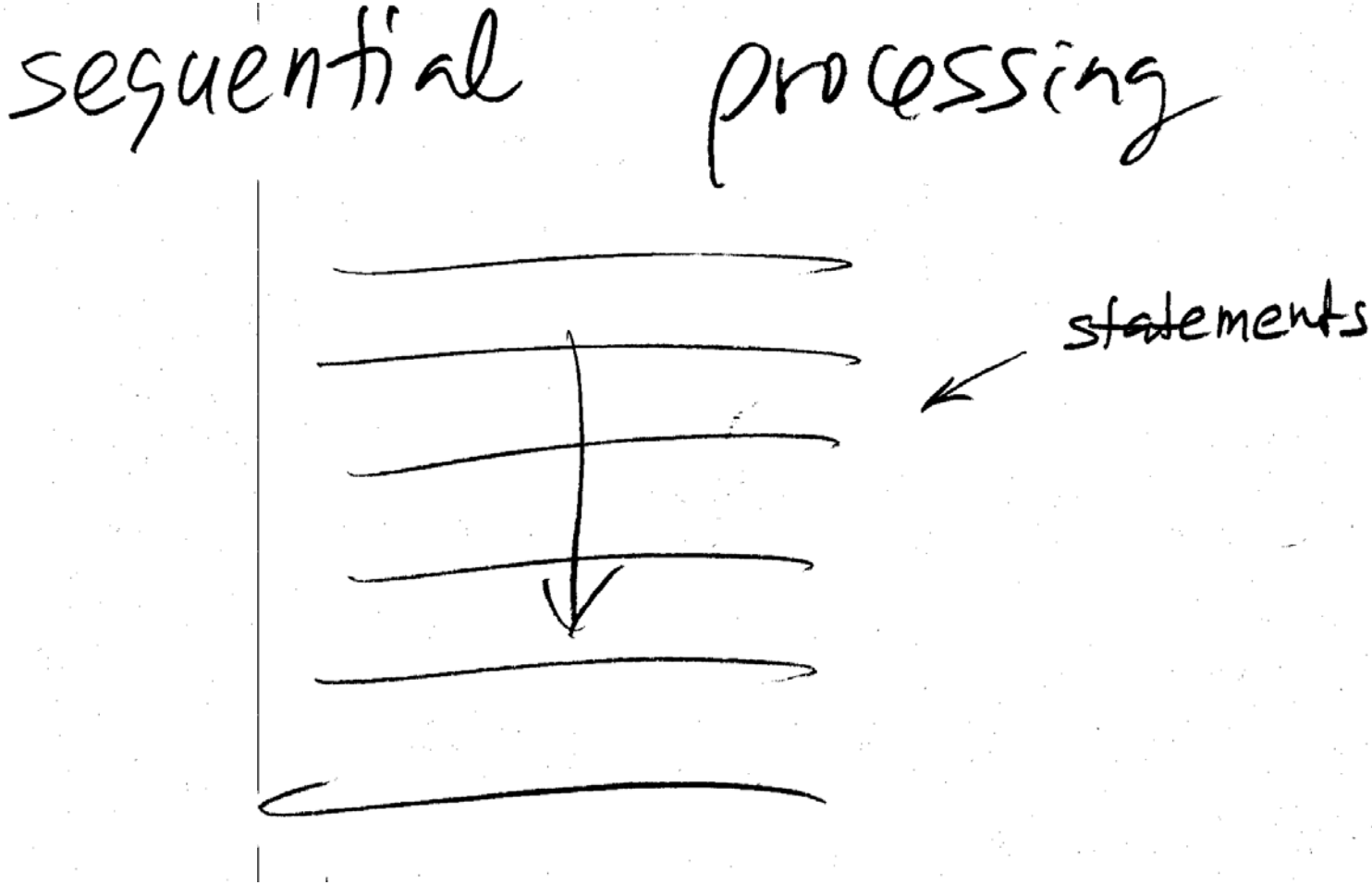
Introducing Logic: IF
and input validation

Valerie Colber, MBA, PMP, SCPM

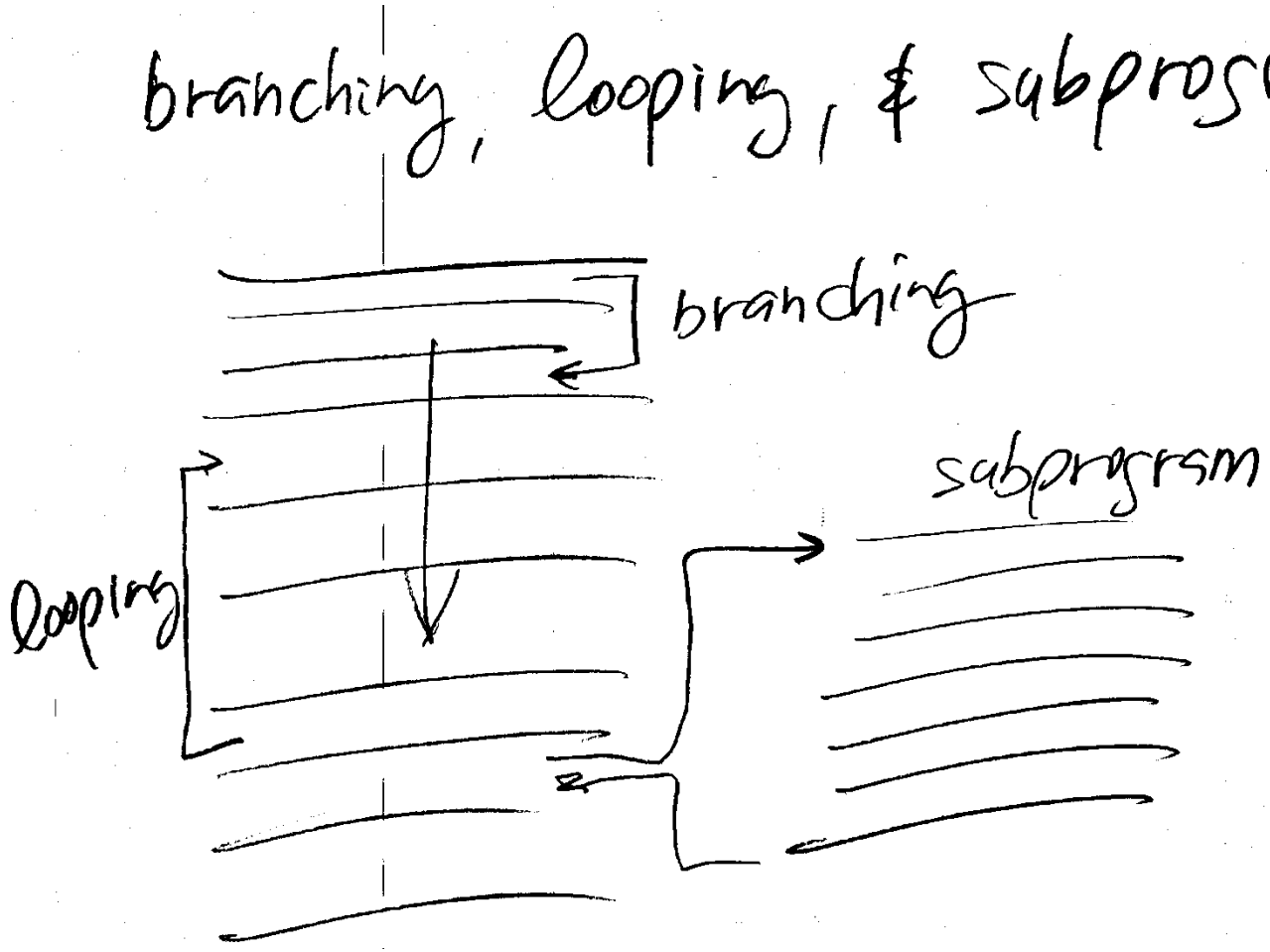
3 control structures

1. Sequence
2. Selection
3. Looping or Repetition

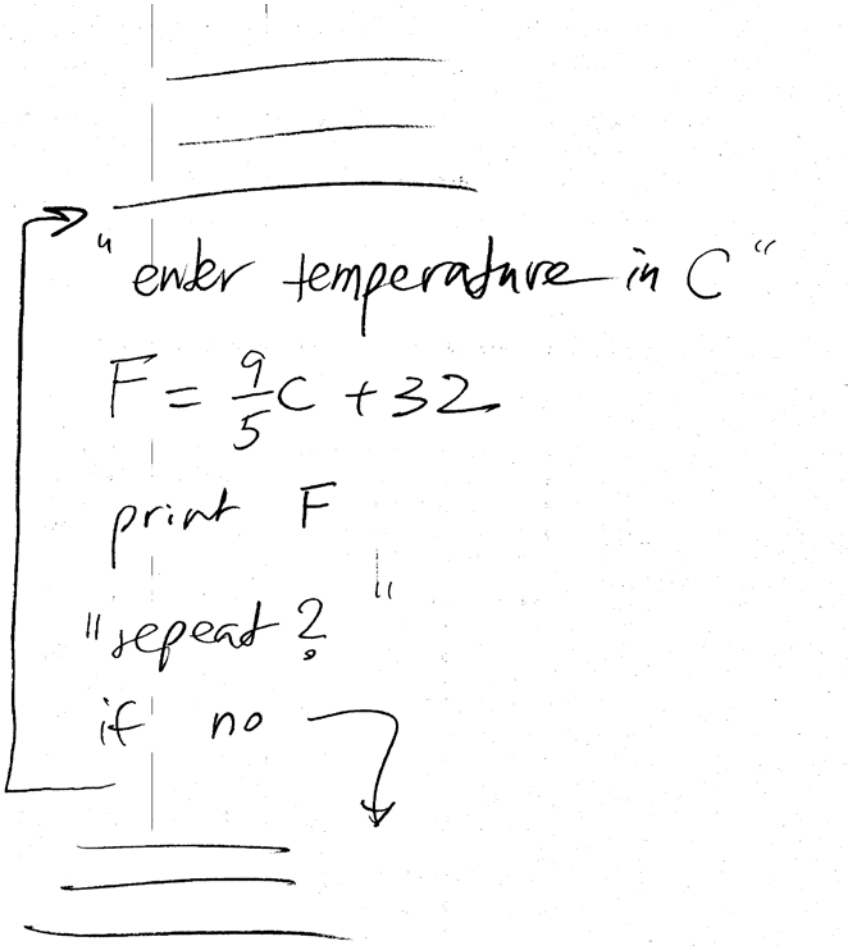
Sequential processing



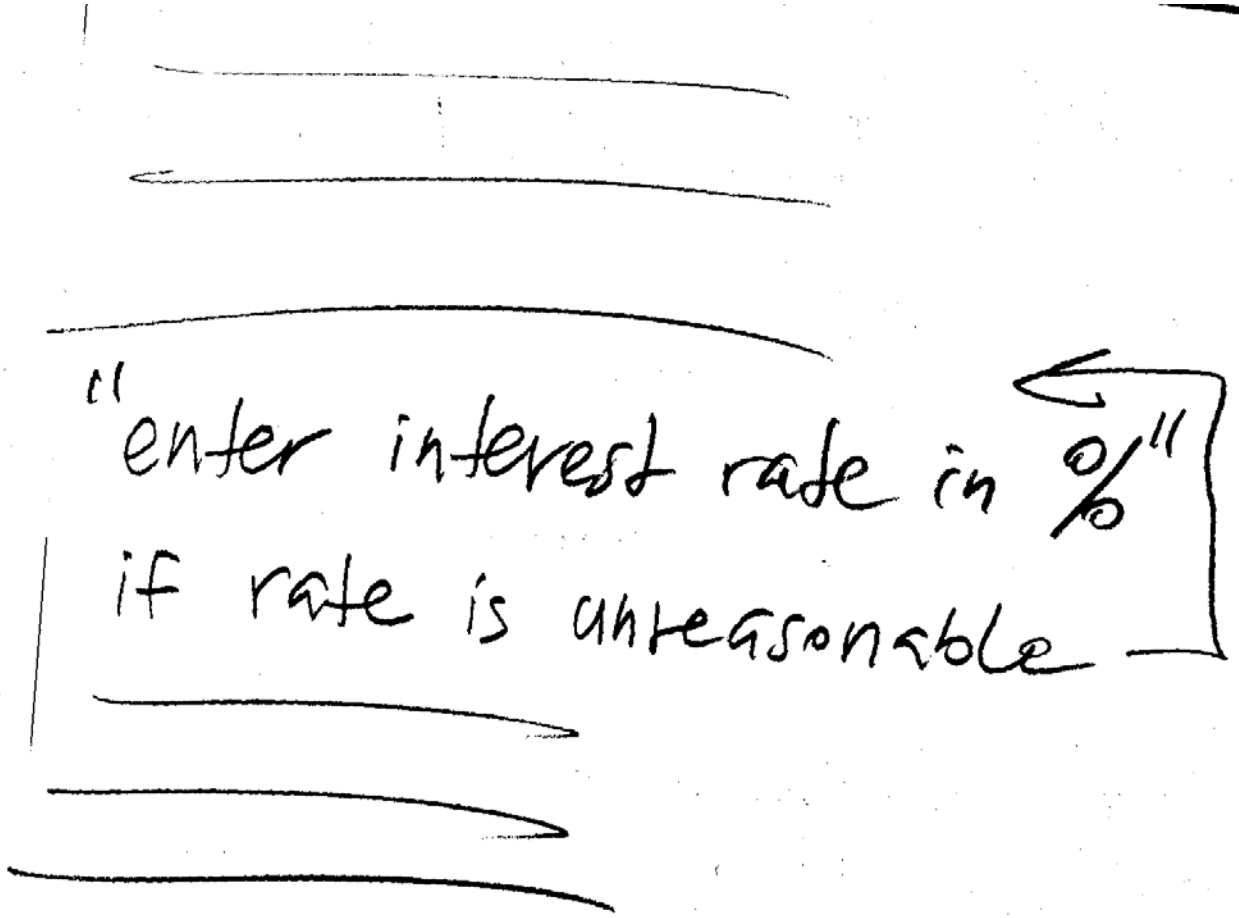
Branching, looping, & subprograms



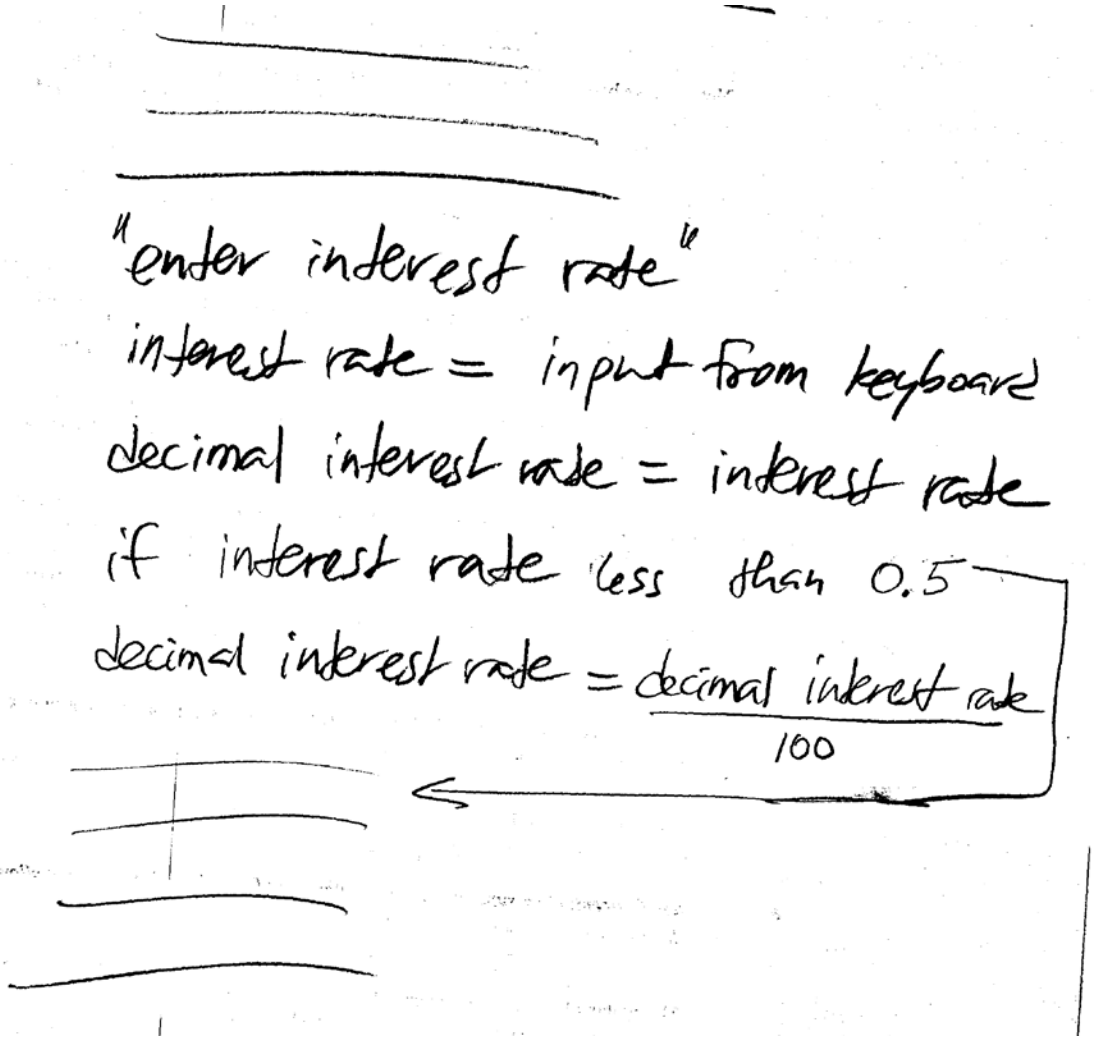
Repeat temperature calculation



Unreasonable interest example



Smart interest rate example



Logic

- By introducing logic in your programs, you can enable them to behave differently depending on the circumstances.
- "If" logic enables programs to test for certain conditions and process the rest of the program accordingly.
- Logic allows you to selectively decide which statements to execute.

Why use logic?

- input validation
- skip calc for invalid input value(s)
- substitute default for invalid input
- …

Conditions and logical expressions

- Besides the **if** statement itself, some other things that you need to know are conditions, logical expressions, and code blocks.
- A condition is something you test for, such as the validity of input data or success in a file opening attempt.
- A logical expression is the C++ code you would use to evaluate a condition.
- These are also called boolean expressions.
- A code block is a series of statements that are used *only* if a tested condition is satisfied.

Conditions

- The easiest way to express conditions in C++ is as something that can be evaluated as true or false.
- There are also multiple choice conditions.

Testing a true or false condition

```
if (name.length() > 0)
{
   cout << “Hi! “ << name << “!” << endl;
}
```

Logical Operators

>	is greater than
<	is less than
>=	is greater than or equal to
<=	is less than or equal to
==	is equal to
!=	is not equal to
&&	and
||	or

Logical expressions

- Note that the symbols for several of these operators consist of *two characters*, which should be typed without being separated by spaces.
- When an operator is typed with two numbers, variables, or expressions on either side of it, the result is a *logical expression*, like **name.length() > 0** .
- There are single spaces separating the operator symbol from the two items it compares -- these are not required, but are there for readability.

Examples of logical expressions

- **x < 10** tests whether the value stored in "x" is less than 10 (x must be an integer or floating point variable)
- **s == "Hello“** tests whether the text stored in "s" is exactly equal to "Hello", with matching case (s must be a **string** variable)
- **a != b** tests whether "a" and "b" have different values (a and b must be variables of the same data type: both integers, both floating points, both characters, or both text)
- **gender == 'F‘** tests whether "gender" is F (gender must be a singlecharacter data type variable, **char**)

Logical operators

- It is important to note that the test for equality has *two* back-to-back equal signs.
- Remember that a single equal sign already means something: assignment of a value to a variable.
- So another symbol was needed for the equality operator.
- Be careful, because it is possible to write **s = "Hello"** (with just one equal sign) as a condition in an **if** statement, but it means something other than a test for equality of content

Comparing text

- Comparisons involving text are done differently for single-character text (**char**) and text of any length (**string**).
- Single-character variables have to be compared using single quote marks around the character (like **'F'**).
- Text of any length needs to be contained inside double quotes (like **"Hello"**).
- Be careful not to mix single-character text with text of any length, because it results in compiler errors.

Complex logical expressions

- Sometimes it is convenient to test for a range of values, or for more than one possible value of a variable.
- For that reason, C++ lets you put multiple conditions together in such a way that the *all* have to be satisfied in order for the "if" code block to be used.
- Here's an example: "if the test score is greater or equal to 80, and less than 90, then assign a grade of 'B'.
- It's also possible to have multiple conditions such that *any one* has to be satisfied in order for the "if" code block to be used.
- Here's an example: "if the grade is 'A', 'B', or 'C', the student passes the class."

Truth Table

True	AND	True	= TRUE
True	AND	False	= FALSE
True	OR	True	= TRUE
True	OR	False	= TRUE

The IF statement

- The code for "skipping" downward the statement involves:
  - a "condition“ *like true/false*
  - "logical" expressions
  - code blocks
- This is what we use instead of "GO TO..."

Programming style

- As a matter of programming style, the curly braces vertically align with the "i" in the word "if", and the code in the code block is indented an additional 2 spaces inside the curly braces.
- This is for readability.
- You will see other ways to write the same thing as you are exposed to code samples from other programmers, in textbooks, and on the Internet.
- There is no right or wrong way among the various styles that programmers use.
- What's important is readability and consistency.
- It would be a good idea for you to follow the programming styles as taught in this class, and to eventually develop your own style.

Programming style

- It is important to note that you cannot place any code between the last (or closing) parenthesis and the first (or opening) curly brace.
- Also, if there is only one statement in the code block, you do not even need the curly braces -- although indenting should still be applied.

classRoster2.cpp

```
//Libraries
#include <iostream>
#include <string>
using namespace std;
…
int main()
{
  //data
   string name; //student name

  //input
  cout << "What is your name? ";
  getline(cin, name);

  // output a greeting, if not blank
  if (name.length() > 0)
  {
    cout << “Hi “ << name <<“!” << endl;
  }//if
}//main
```

Example algorithm using AND logic

- read a score from the keyboard as a whole number
- if the score is 80 or greater AND less than 90
- print "grade B"

gradeB.cpp

```
//Libraries
#include <iostream>
using namespace std;
…
int main()
{
  //data
    int score; //user score

  // input a score from the keyboard
  cout << "What is your test score? [0-100]: ";
  cin >> score;
  cin.ignore(1000, 10);

  // see if score is in the B range
  if (score >= 80 && score < 90)
     cout << "Your grade is B" << endl;
}//main
```

Notes on gradeB.cpp

- The symbol that joins two (or more) logical expressions into a complex "and" expression is **&&** .
- Note the single spaces surrounding the symbol -- they are not required, but are there for readability.
- Also note that there is repetition in the complex statement -- **score** is typed twice.
- There is no way to say "if the score is greater than or equal to 80 and less than 90" in C++.
- You have to say "if the score is greater than or equal to 80 *and* the score less than 90" instead.
- That is, complex logical expressions can only be built by joining simple ones.

Notes on gradeB.cpp

- Also note that the code block associated with the if does not need to be contained in curly braces when there is only one statement.
- Further, that statement can even by typed on the same line as the **if**.
- It might not make much difference to you now, but C++ does not always need to evaluate all of the joined logical expressions.
- For example, if the score is 50, the "greater than or equal to 80" part is false, so there is no point in checking if "the score less than 90".

Algorithm using OR logic

- input a grade from the keyboard as a single letter
- if the grade is uppercase A, B, or C
- output "you pass"

passingGrade.cpp

```
//Libraries
#include <iostream>
using namespace std;
…
int main()
{
  //data
  char grade; //user’s grade

  // input a grade from the keyboard
  cout << "What is your grade? [A, B, C, D, or F]: ";
  cin >> grade;
  cin.ignore(1000, 10);

  // check for passing grade
  if (grade == 'A' || grade == 'B' || grade == 'C’)
  {
  cout << "You pass" << endl;
  }//if
}//main
```

Notes on passingGrade.cpp

- The symbol that joins two (or more) logical expressions into a complex "or" expression is **||** .
- This is not commonly typed symbol outside of programming -- it's usually the SHIFT-BACKSLASH keystroke.
- Note the single spaces surrounding the symbol -- they are not required, but are there for readability.
- Also note that there is repetition in the complex statement -- **grade** is typed three times.
- You do not say "if the grade is A or B or C" in C++.
- You say "if the grade is A *or* the grade is B *or* the grade is C" instead.
- Remember that complex logical expressions can only be built by joining simple ones.

Notes on passingGrade.cpp

- Again, C++ does not always need to evaluate all of the joined logical expressions.
- For example, if the grade is A, the "grade is A" part is true, so there is no point in checking if the "grade is B" or if the "grade is C".
- Be careful -- it is possible to join logical expressions with **&&** s and **||** s in the same complex expression.
- But it is usually not a good idea, because the exact meaning of the logic can be confusing.
- It has to do with the order in which operations are evaluated. For example, don't do **score >= 0 && score < 80 || score >= 90 && score <= 100** to test for scores that are not B.
- But you could do **(score >= 0 && score < 80) || (score >= 90 && score <= 100)** instead!
- Also note that the variable that appears multiple times in complex logical expressions does not have to be the same variable in each of the simple expressions.
- You can do this, for example: **score > 86 && grade == 'B'** to test for a high B.

Opposite Logic versus Not Logic

- Sometimes it is easier to define the scope of what you do not want or what is not included because it is less than
- DeMorgan's Theorum is Opposite Logic which means you reverse all logical operators for the opposite logical operator.
- Not Logic means "not this" or "anything else but this".

DeMorgan Theorem

DeMorgan Theorem Samples

Here are samples of logical expressions, and their opposites, applying two techniques: (1) DeMorgan's theorem and (2) "not" logic:

Original Expression	It's Opposite, Applying DeMorgan's Theorem	It's Opposite, Applying "Not" Logic
`if (age == 21)` `  cout << "You are 21" << endl;`	`if (age != 21)` `  cout << "You are not 21" << endl;`	`if (!(age == 21))` `  cout << "You are not 21" << endl;`
`if (score >= 90)` `  cout << "Your grade is A" << endl;`	`if (score < 90)` `  cout << "Your grade is not A" << endl;`	`if (!(score >= 90))` `  cout << "Your grade is not A" << endl;`
`if (gender == 'M' || gender == 'm')` `  cout << "You are male" << endl;`	`if (gender != 'M' && gender != 'm')` `  cout << "You are not male" << endl;`	`if (!(gender == 'M' || gender == 'm'))` `  cout << "You are not male" << endl;`
`if (!fout.good())` `  cout << "The file cannot be created" << endl;`		`if (fout.good())` `  cout << "The file can be created" << endl;`
`if (score >= 80 && score < 90)` `  cout << "Your grade is B" << endl;`	`if (score < 80 || score >= 90)` `  cout << "Your grade is not B" << endl;`	`if (!(score >= 80 && score < 90))` `  cout << "Your grade is not B" << endl;`

Handling case

- What if the prompt for grade tells the user to enter an uppercase letter grade, but the user types a lowercase letter instead?
- After all, the letters are uppercase on the keyboard!
- Rather than not recognize 'a' as 'A', you can do one of two things: check for either case, or convert case before checking the value.
- For example:

grade == 'A' || grade == 'a'

or

toupper(grade) == 'A'

Or Else

- Often it is convenient to use logic to switch between two possible code blocks.
- This is done with the **else** "clause" to the **if** statement.

Algorithm using Else logic

- input a grade from the keyboard as a single letter
- if the grade is uppercase A, B, or C, then output "you pass"
- Otherwise output "you don't pass"

passNoPass.cpp

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

  // input a grade 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;
  }
  else
  {
    cout << "You do not pass" << endl;
  }//if
}//main
```

Notes on passNoPass.cpp

- The **else** clause is followed by another code block to be used if the logical expression in the **if** statement's condition evaluates to false.
- Its code block needs to be enclosed in curly braces if it involves more than one statement.

Multiple choice with IF algorithm

- input user's grade via keyboard
- if A, output "excellent"
- otherwise if B, output "good"
- otherwise if C, output "average"
- otherwise if D or F, output "see ya..."
- otherwise, output "invalid"

grades.cpp

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

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

  //check grade
  if (grade == 'A')
    cout << "Excellent" << endl;
  else if (grade == 'B')
    cout << "Good" << endl;
  else if (grade == 'C')
    cout << "Average" << endl;
  else if (grade == 'D' || grade == 'F')
    cout << "See you next year" << endl;
  else
    cout << "Invalid: " << grade << endl;
}//main
```

IF rules

- "If"s can have zero, one, two, or more **else if** clauses.
- Once an **if** or **else if** is found to be true, no further checking of any remaining **else if** clauses happens.
- There can be zero or one **else** clause, which must come after any **else if** clauses.
- Each clause has its own code block, which must be enclosed in curly braces if there are multiple statements in the block.
- Curly braces are legal but not required for single-statement code blocks.

Alternative code for multiple choice

- A drawback of the if-elseif-else structure is the requirement to repeat the tested variable over and over again.
- C++ provides another way to represent the same logic without the repeats -- the **switch** statement.

gradeSwitch.cpp

```
//Libraries
#include <iostream>
using namespace std;
…
int main()
{
  //data
  char grade;
  cout << "What is your grade? [A, B, C, D, or F]: ";
  cin >> grade;
  cin.ignore(1000, 10);
  switch (grade)
  { // start here with switch OFF
    case 'A': // if grade is 'A', turn switch ON
    case 'a': // if grade is 'a', turn switch ON
       cout << "Excellent" << endl; // do this if switch is ON
       break; // skip to closing curly brace if switch is ON
    case 'B': // if grade is 'B', turn switch ON
    case 'b': // if grade is 'b', turn switch ON
       cout << "Good" << endl; // do this if switch is ON
       break; // skip to closing curly brace if switch is ON
    case 'C': // if grade is 'C', turn switch ON
    case 'c': // if grade is 'c', turn switch ON
       cout << "Average" << endl; // do this if switch is ON
       break; // skip to closing curly brace if switch is ON
    case 'D': // if grade is 'D', turn switch ON
    case 'F': // if grade is 'F', turn switch ON
    case 'd': // if grade is 'd', turn switch ON
    case 'f': // if grade is 'f', turn switch ON
       cout << "See you next year" << endl; // do this if switch is ON
       break; // skip to closing curly brace if switch is ON
    default: // turn switch ON
       cout << "Invalid: " << grade << endl; // do this if switch is ON
  } // switch
}//main
```

Notes on gradeSwitch.cpp

- "Switch" can be used with integers or single-character text only -- not floating point numbers or text of any size.
- The **break;** statement marks the end of a code block -- curly braces are not required for the separate code blocks, even if they have more than one statement!
- The **default:** condition works like the **else** clause to the "if" statement -- it is not required, and does not need to end in **break;** .

Notes on gradeSwitch.cpp

- When using "switch" with integers, use statements like **case 0:** , without quote marks around the tested value.
- But when using "switch" with single-character text, use statements like **case 'A':** , *with* single quote marks around the tested value.
- "Switch" statements cannot have ranges or variables in the **case** statements.

Notes on gradeSwitch.cpp

- The **switch** statement gets its name from the fact that it is like an on/off switch.
- It starts in the "off" position, and checks each **case** statement from top to bottom until it finds a match with the value in its parentheses.
- When (and if) a match is found, or if the **default** statement is reached, the switch is set to "on".
- From then on, all other **case** statements are ignored, and any code that appears is executed.
- The purpose of **break;** is to prevent the execution from going any further, and running into and over the next **case** -- it skips to the end of the **switch** statement, after its closing curly brace.

Comparing two numbers algorithm

- input the 1st number from the keyboard
- input the 2nd number from the keyboard
- if the 1st number is smaller than the 2nd then output that it is so
- otherwise if the 2nd number is smaller than the 1st then output that it is so
- Otherwise output that the numbers are the same

compare2Numbers.cpp

```
//Libraries
#include <iostream>
using namespace std;
…
int main()
{
   //data
   int a;  //first number
   int b;  //second number

   //input
   cout << "Enter the 1st number: ";
   cin >> a;
   cin.ignore(1000, 10);
  cout << "Enter the 2nd number: ";
   cin >> b;
   cin.ignore(1000, 10);

   //compare
   if (a < b)
      cout << a << " is less than " << b << endl;
   else if (b < a)
      cout << b << " is less than " << a << endl;
   else
      cout << a << " is the same as " << b << endl;
}//main
```

A quiz algorithm

- prompt the user with the quiz question for 8 + 2
- input the user's answer from the keyboard
- if the answer is 10 then output "correct"
- Otherwise output "incorrect"

quiz.cpp

```
//Libraries
#include <iostream>
using namespace std;
…
int main()
{
   //data
   int answer;  //user answer to math problem

   // input
   cout << endl << endl << endl << endl;
   cout << " 8" << endl;
   cout << " +2" << endl;
   cout << " --" << endl;
   cout << " ";
   cin >> answer;
   cin.ignore(1000, 10);
   cout << " ";

  //output feedback
   if (answer == 10)
      cout << "Correct!" << endl;
   else
      cout << "Very good, but a better answer is 10" << endl;
}//main
```

Classic computer science solutions

- Two classic problems are finding the lowest (or highest) from among a set of values, and sorting a set of value from lowest to highest (or the reverse).
- The solutions involve the logic of **if** statements.
- They also involve techniques that will be covered in chapters 8 and 11, so while we cannot develop full solutions here, we can at least show the logical parts of the solutions.

Finding The Maximum Or Minimum Value

```
// get values for 3 integers
int a = ...
int b = ...
int c = ...
```

```
// find the maximum integer
int max = a;
if (max < b) max = b;
if (max < c) max = c;
```

```
// find the minimum integer
int min = a;
if (min > b) min = b;
if (min > c) min = c;
```

Finding the minimum or maximum value

- You may not have come up with this logic on your own right away, because it is somewhat counterintuitive.
- For example, note that there are variables for storing the minimum and maximum values, but they are set to the value of *the first number*!
- It's as if we know the answer in advance, and the answer is the first number -- we'd have to be pretty lucky for this to work!
- But the situation is that our logic statements can only compare 2 values at a time, and we have 3.
- So here's the approach: without looking at any of the numbers, *assume* that the first number is what we want.
- Then look at the second number, and if it is better, we change our mind!
- Then look at the third number, and if *it* is better, we change our mind, perhaps again.
- So you can probably figure out how to extend this to 4 or more values -- just add more **if** statements.

Another classic problem in computer science is reordering multiple values from lowest to highest (or the reverse).

The sets of three lines of code, starting with declaration of **temp** , is called "swap code" -- it swaps the values of 2 variables.

Sorting Values Lo-to-Hi

```
// get values for 3 integers
int a = ...
int b = ...
int c = ...
```

```
// move the lowest value to "a"
if (a > b)
{
  int temp = a;
  a = b;
  b = temp;
}
if (a > c)
{
  int temp = a;
  a = c;
  c = temp;
}
```

```
// move the next lowest value to "b"
if (b > c)
{
  int temp = b;
  b = c;
  c = temp;
}
```

```
// ...the highest value is now in "c"
```

The complete IF statement

```
if (condition)
{
  …
}
else if (condition)
{
  …
}
else
{
  …
}//if
```

OPTIONAL: 0-many else if (condition) with statement(s) in { } if condition is true

OPTIONAL: zero or only 1 else with statement(s) in { } if the original if (condition) was false and any or all else if (condition(s)) was false, so this is the default what to do.

4 forms of the basic IF statement

1. "if" With Curly Brace Code Block

- The most common example has a multi-statement code block contained in curly braces.
- When the condition in the parentheses of the **if** statement evaluates to true, the code block is not skipped.

```
if (...)
{
...
... one or more
... statements
...
}
```

4 forms of the basic IF statement

2. "if" Without Curly Brace Code Block

- Another common example has a single-statement code block without curly braces.
- When the condition in the parentheses of the **if** statement evaluates to true, the statement is not skipped.
- This offers a more condensed way to write code, without doubling the amount of vertical space by including curly braces.

if (...)
***statement*;**

4 forms of the basic IF statement

3. "if" All On One Line

- A single-statement code block can also follow the **if** on the same line, condensing the code even further.
- This is suitable for very short statements.

if (...) ***statement*****;**

4 forms of the basic IF statement

4. Beware! "if" With Semicolon

- There is no good reason to use this form, but it is valid, and your program will compile if you use it.
- The semicolon after the condition's parentheses effectively cuts off the **if** statement from whatever follows it -- indented or not!
- The only reason for showing this form is to make you aware of this pitfall.
- It is a difficult programming error to diagnose, because the compiler accepts it and it leads to logically incorrect results when the program runs.

if (...);

An IF that doesn't perform as expected

- This prints "excellent" for *any* value of **grade** -- not just **'A'** .
- The reason is that the semicolon at the end of the "if" statement *ends* the statement!
- Even though it is indented, the print statement does *not* belong to the "if".
- It is a separate statement, as if it was not indented at all.
- But it does compile, and that is the unfortunate part -- C++ allows you to do this even though you probably would prefer that it gave a compilation error.
- There is one exception -- if you add an **else** or **else if** clause, that clause will be seen as standing alone without an "if", and compilers won't allow that!

```
if (grade == 'A');
   cout << "excellent" << endl;
```

WARNING: This preview/introduction prior to chapter 7 of the repetition control structure only introduces one of four forms of the repetition control structure which will be fully covered in chapter 7. DO NOT become reliant on just this first form of the repetition control structure in C++. You will need to understand WHEN to use each of the 4 forms of the repetition after we cover Chapter 7.

Repetition or looping

Introduction of Chapter 7 so we can begin doing input validation

Kinds of loops

- Event controlled
- Count controlled

Syntax: The [hardcoded] 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

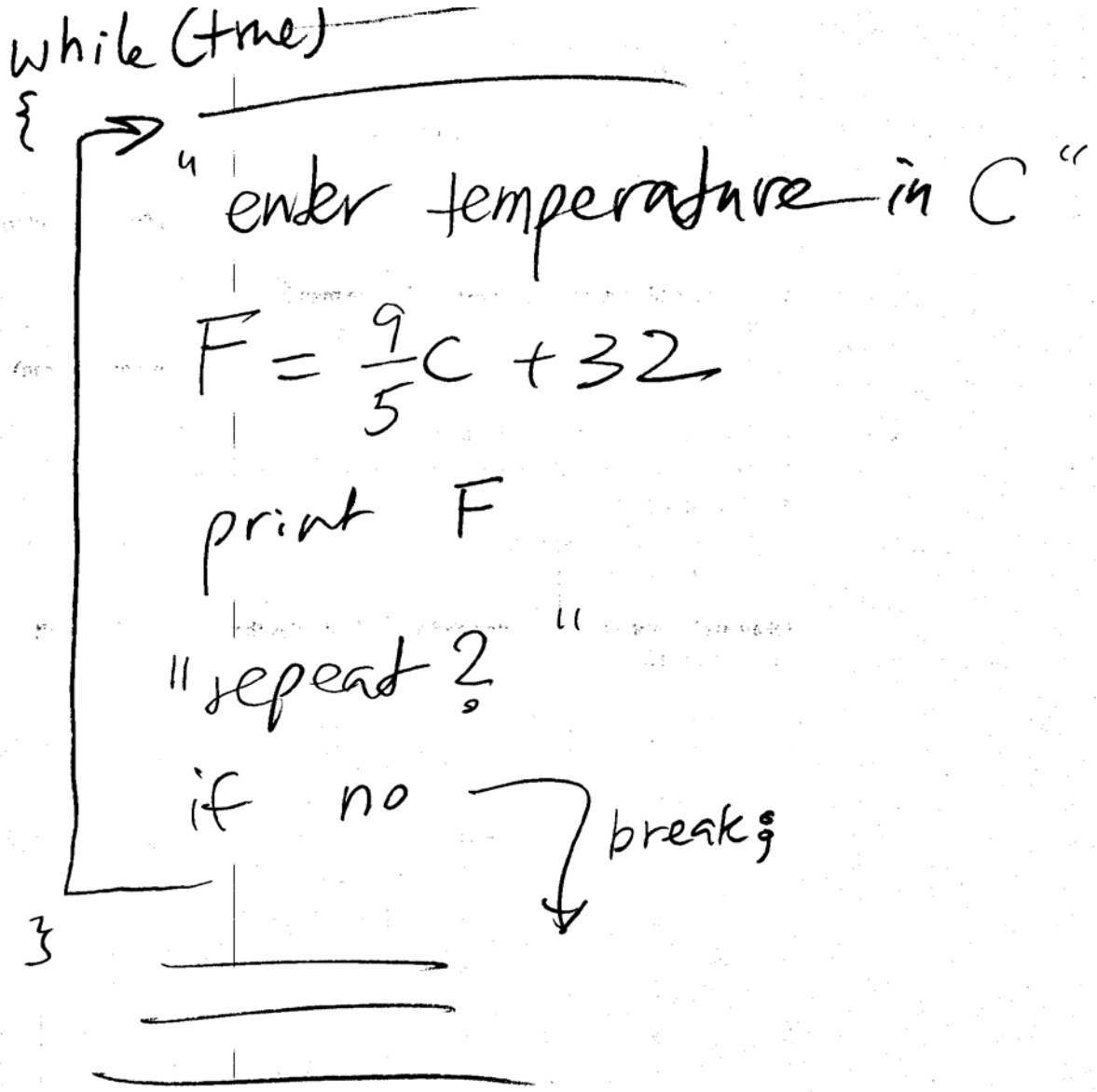
Scope

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

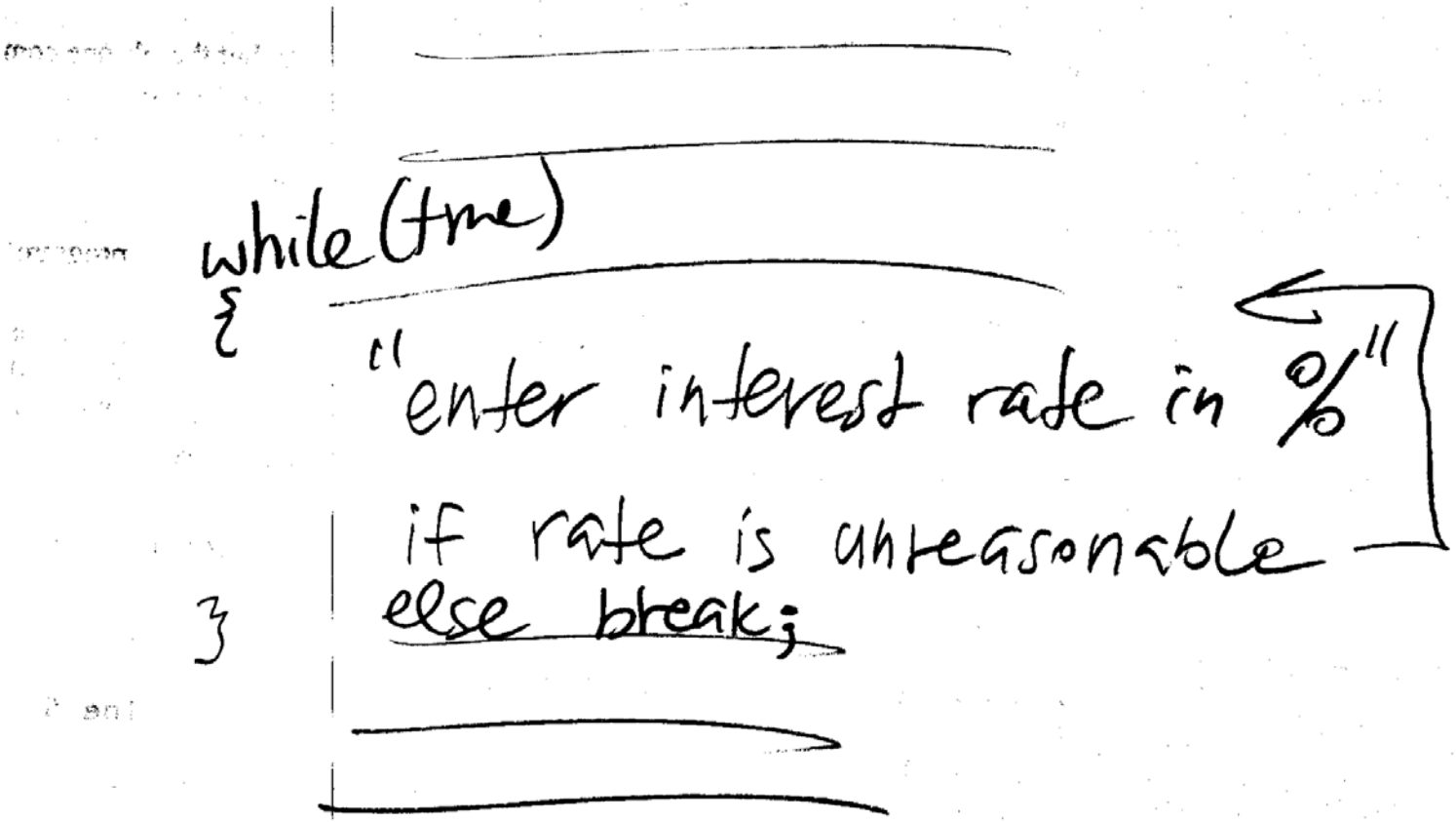
```
{
   ...
   int a;
   "a" is useable here...
}
...but not here
```

This is why we begin the main program with the data code block, and declare/define all data used in the program FIRST!

Repeat temperature calculation



Unreasonable interest rate



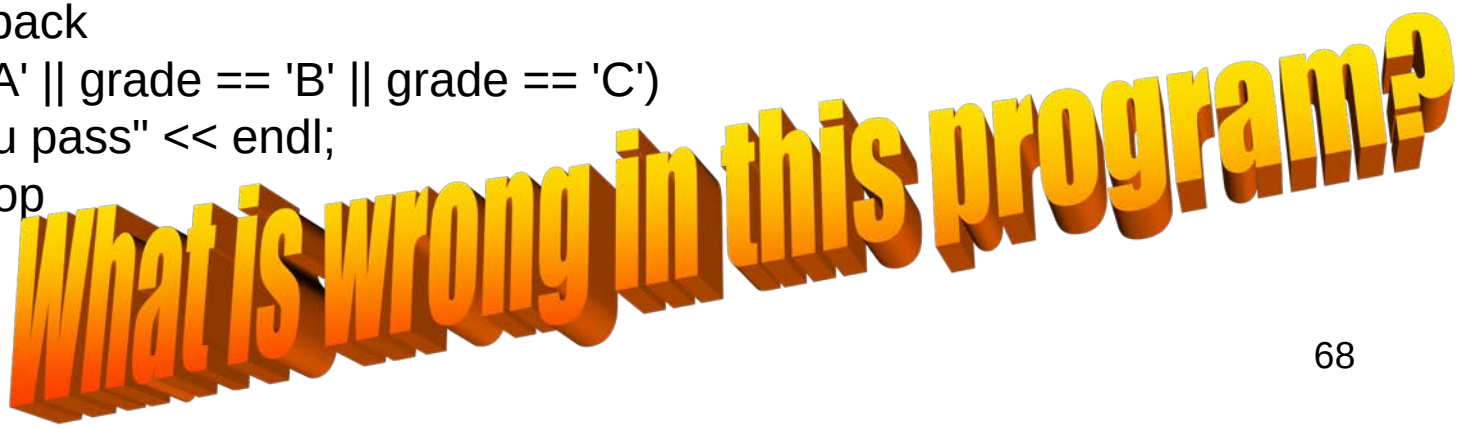
passingGradeL.cpp

```
#include <iostream>
using namespace std;
…
int main()
{
  while (true)
  {
    //data
    char grade; // read a char value from the keyboard

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

    //output feedback
    if (grade == 'A' || grade == 'B' || grade == 'C')
      cout << "You pass" << endl;
  }//while true loop

}//main
```



What is wrong with passingGradeL.cpp?

- This loops over and over, actually never ending!
- The only way out of this loop is to press CTRL-C instead of typing a grade when prompted!
- There really should be a prompt 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.

Be careful where you put stuff in your program

- variable **grade** is declared "above/before/outside the loop".
- It could just as well have been declared "inside the loop", and there is there is no syntax error.
- If still used, **return 0;** statement is "below the loop".
- If it was to be moved inside the loop, the program would terminate before reaching the end of the loop the first time through.

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;
...
}
```

passingGradeIB

```
#include <iostream>
using namespace std;
…
int main()
{
  while (true)
  {
     char grade;
     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
```

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 Controlled Loops

```
...
//data
string answer; //user response

//input validation
while (true)
{
  //input
  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; //user answer Y or N

  //input validation
  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
```

Adding logic to previous programs

ItsAboutYou

```
#include <iostream>
#include <string>
using namespace std;
…
int main()
{
  //data
  int age;
  double gpa;
  string name;
  char mInitial;

  //input
  cout << "What is your age? ";
  cin >> age;
  cin.ignore(1000, 10);
  cout << "What is your grade point average? ";
  cin >> gpa;
  cin.ignore(1000, 10);
  cout << "What is your name? ";
  getline(cin, name);
  cout << "What is your middle initial? [A-Z]: ";
  cin >> gender;
  cin.ignore (1000, 10);
  cout << "Your age is " << age << endl;
```

```
  //input validation
  if (age >= 30)
  {
    cout << “You are probably good at this!" << endl;
  }
  cout << "Your GPA is " << gpa << endl;
  if (gpa > 3.6)
  {
    cout << “Are you transferring to UC Berkeley?" << endl;
  }
  cout << "Your name is " << name << endl;
  if (name.length() == 0)
  {
    cout << "So, you have no name?" << endl;
  }
  if (mInitial >= ‘A’ || mInitial <= ‘Z’)
  cout << "Your middle initial is " << mInitial << endl;
  else cout << “invalid middle initial…” << endl;
}//main
```

Lighthouse

```
#include <iomanip>
#include <iostream>
using namespace std;
#include <cmath>
…
int main()
{
  //data
  double height; //height of lighthouse in feet
  cout << "Enter lighthouse height, feet: ";
  cin >> height;
  cin.ignore(1000, 10);

  //input validation/ output result
  if (height > 0.0)
  {
    double distance = sqrt(0.8 * height);
    cout << fixed;
    cout << setprecision(0);
    cout << "A " << height << "-foot tall lighthouse can be seen for ";
    cout << distance << " miles." << endl;
  }//if
}//main
```

Scope issues

- The following samples show the concept of "scope" as it relates to variable declarations and curly brace containers.
- They are based on the lighthouse exercise.
- The logic change in the program is to proceed with the calculation if height is zero or negative, but to use zero for the height even if it's entered as a negative number.

Original Program

```
#include <iomanip>
#include <iostream>
using namespace std;
#include <cmath>
…
int main()
{
  //data
  double height; //height of lighthouse in feet
  double distance; //distance lighthouse can be seen from ships to shore in miles

  //input
  cout << "Enter lighthouse height, feet: ";
  cin >> height;
  cin.ignore(1000, 10);

  //input validation/ output result
  if (height > 0.0)
  {
    distance = sqrt(0.8 * height);
    cout << fixed << setprecision(0);
    cout << "A " << height << "-foot tall lighthouse can be seen for ";
    cout << distance << " miles." << endl;
  }
  else cout << "Sorry, you entered a height that was not greater than zero." << endl;
}//main
```

```
#include <iomanip>
#include <iostream>
using namespace std;
#include <cmath>
…
int main()
{
  //data
  double height; //height of lighthouse in feet

  //input
  cout << "Enter lighthouse height, feet: ";
  cin >> height;
  cin.ignore(1000, 10);

  //input validation/ output result
  if (height > 0.0)
  {
    double distance= sqrt(0.8 * height); // "distance" declared here
  } // "distance" goes out of scope at 1st } to its left
  else
  {
    height = 0.0; // in case it was negative
    double distance = 0.0; // another "distance" declared here
  } // "distance" goes out of scope at 1st } to its left
  // COMPILER ERROR: neither "distance" can be used: both are out of scope
  cout << fixed << setprecision(0);
  cout << "A " << height << "-foot tall lighthouse can be seen for ";
  cout << distance << " miles." << endl;
}//main
```

Incorrect version
If you did not use
Data code block

This version works If you are not required to use a Data code block

```
#include <iomanip>
#include <iostream>
using namespace std;
#include <cmath>
…
int main()
{
  double height;
  cout << "Enter lighthouse height, feet: ";
  cin >> height;
  cin.ignore(1000, 10);
  double distance; // declare here so it's still in scope after this code block
  if (height > 0.0)
  {
    distance = sqrt(0.8 * height); // "distance" assigned here
  }
  else
  {
    height = 0.0; // in case it was negative
    distance = 0.0; // "distance" assigned here
  }//if
  cout << fixed << setprecision(0);
  cout << "A " << height << "-foot tall lighthouse can be seen for ";
  cout << distance << " miles." << endl
}//main
```

Another correct version

```
#include <iomanip>
#include <iostream>
using namespace std;
#include <cmath>
…
int main()
{
  //data
  double height; //height of lighthouse in feet
  double distance = 0.0; //distance ships can see the lighthouse from shore in miles

  //input
  cout << "Enter lighthouse height, feet: ";
  cin >> height;
  cin.ignore(1000, 10);
  if (height > 0.0)
  {
    distance = sqrt(0.8 * height);
  }
  else
  {
    height = 0.0; // in case it was negative
  }//if
  //output
  cout << fixed << setprecision(0);
  cout << "A " << height << "-foot tall lighthouse can be seen for " << distance << " miles." << endl;
}//main
```

Another correct version

```
#include <iomanip>
#include <iostream>
using namespace std;
#include <cmath>
…
int main()
{
  //data
  double height; //height of lighthouse in feet
  double distance; //distance ships can see the lighthouse from shore in miles

  //input
  cout << "Enter lighthouse height, feet: ";
  cin >> height;
  cin.ignore(1000, 10);

  //input validation /calculate distance
 if (height < 0.0) height = 0.0; // a 1-line code block, on same line as "if"
 distance = sqrt(0.8 * height);

  //output
  cout << fixed << setprecision(0);
  cout << "A " << height << "-foot tall lighthouse can be seen for ";
  cout << distance << " miles." << endl;
}//main
```

References

- INTRODUCTION TO PROGRAMMING USING C++ by Robert Burns