C++ Fall 2019 Notes & Documentation:

Basics:

* #include <iostream> 🡪 allows you to read & write standard the standard output streams
* using namespace std; 🡪 allows us to use objects and variables from the standard library
* int main() {} 🡪 where everything is executed
* #include “pch.h” 🡪 ??
* Return 0; 🡪 ??

Variables/Data Types:

* Int
* Double
  + Float vs double?
  + Float – allows 7 digits after the decimal
  + Double – allows 15 digits after the decimal
* Char
* String
  + #include <string>
  + string greeting = “Hello”;
  + cout greeting[0]; //outputs H
* Bool
* Const 🡪 read only

Input/Output:

* cout 🡪 display to the screen – uses insertion operator <<
* cin 🡪 get user input – uses extraction operator >>
* Example:
  + string firstName;
  + cout << “What is your name?”;
  + cin >> firstName;
  + cout << “Your name is:” << firstName;
* \*cin considers a space/tab as a terminating character
  + string fullName;
  + cout << “What is your full name?”;
  + getline(cin, fullName);
  + cout << “Your full name is: ” << fullName;

Binary Operators: operate on 2 operands 🡪 \* / % + -

Unary Operators: operate on only 1 operand 🡪 - 🡪 flips sign of current value?

**Chapter 3**:

The CIN Statement:

* Cin automatically converts the data read from the keyboard to the data type of the variable used to store it
* Requires <iostream> header file, enter key is pressed after data is entered
* Can be used to gather multiple values (separated by spaces)
* EX:

Arithmetic operators and their precedence:

* When 2 operators share an operand, the operator with the highest precedence works first.
* Highest 🡪 \* / %
* Lowest 🡪 + -

Set Precision Statement:

* Sets the number of significant digits
* Specify where floating point will go, counting from right
* Stream manipulator

Fixed Manipulator:

* stream manipulator
* same as setprecision, with force decimal notation
* must be used with setprecision???

ShowPoint:

* adds zeros to equate to the number of places specified

The **>>, <<** operators:

* >> 🡪 stream extraction operator: gets characters from the stream object on its left, and stores them in the variable on its right
* In a statement that uses COUT – the << operator always points towards cout. This indicates that data is flowing from a variable or a literal to the cout object.
* In a statement that uses CIN – the >> operator always points towards the variable that is receiving the value. This indicates that the variable

The header file <iostream>:

* Ok

Associativity:

* the order in which an operator works with its operands
* An operator’s associativity is either: left 🡪 right OR right 🡪 left
* Unary negation - 🡪 right to left
* \* / % 🡪 left to right
* + - 🡪 left to right

When C++ is working with an operator, it strives to convert operands to the same type. This is known as:

* Automatic type conversion **🡪 Type Coercion**
* Promoted: when a value is converted to a higher data type
* Demoted: when a value is converted to a lower data type

When a variable is assigned a number that is too large for its data type, what happens?

* The variable overflows
* Attempting to store a number too large for the given variable type

setw manipulator:

* specifies the field width for the value immediately following it
* the field width is the minimum number of character positions, or spaces on the screen to print the value in it.
* Cout << “(” << setw(5) << value << “)”; // value = 23
* ( 23) //output
* Requires <iomanip>

The function **pow(x, y)**, requires which header file?

* <cmath>
* Unlike many programming languages, C++ does NOT contain an exponent operator
* This requires a library function

Rand() Function:

* Needs header file – <cstdlib>

Hand Tracing:

I will give you snippets of code and you should be able to identify the answer. Here is an example.

Which of the following will allow the user to input the values **15** and **20** and have them stored in variables named **base** and **height**, respectively?

|  |  |
| --- | --- |
| a. | **cin << base << height;** |
| b. | **cin base, height;** |
| c. | **cin >> base >> height;** |
| d. | **cin base >> cin height;** |
| e. | None of these |

**Chapter 4: Making Decisions**:

Relational Operators:

* ==, >=, <=, !=
* Used to compare numeric data
* All relational operators are binary – need to operands to operate on
* All relational operators have left to right associativity
* Each relational operator determines whether a specific relationship exists between 2 values
* Relational Expressions are also known as Boolean expressions – value can only be true or false

(ex: x > y 🡪 used to determine if x is less than y)

Logical Operators:

* Connects two or more relational expressions into one, or reverse the logic of an expression
* Typically used with Boolean values
* && (logical AND) – takes 2 expressions as operands and creates an expression that is true only when both subexpressions are true.
* || (logical OR) – takes 2 expressions as operands and creates an expression that is true when either of the subexpressions are true. \*performs short-circuit evaluation
* ! (logical NOT) – takes 1 operand and reverses its truth or falsehood. Must first evaluate the expression, then apply the “!” – flop the value.

\*C++ does not allow you to check numeric ranges with expressions such as: 5 < x < 20. Instead you must use a logical operator to connect the two relational expressions: x >= 5 && x <= 20.

sequence structure: statements are executed in a sequence, without branching off in another direction

decision structure: a specific action is taken only when a specific condition exists

If Statement:

* if the value of the expression inside of the parenthesis is true, the very next statement/group of statements inside the containing brackets is/are executed. Otherwise, the statement(s) is skipped.
* Braces needed if more that one statement should be executed upon true condition/expression

Comparing floating point numbers?

If/Else Statement:

* Will execute one group of statements if the expression is true, and a different group of statements if the expression is false.

Nested If Statement:

* Used to test multiple conditions, separately.
* Could also be done by using if/else if statements
* Could also be done by combining conditions & using logical operators in same expression.

If/Else If Statement:

* Tests a series of conditions, like nested if statements.
* Often a simpler way to test multiple conditions than using nested ifs.
* Each expression is tested sequentially, until a true evaluation is found. All other expressions/statements are skipped and the if statement is over.
* If none of the expressions are true, then the last else clause is executed.

Flag:

* A Boolean or integer variable that signals when a condition exists
* If the flag variable is set to true – the condition exists
* If the flag variable is set to false – the condition does not exist
* Must set the flag variable based on a certain condition, then you can later test that variable and use it to determine a conditionally executed statement.

(ex: if (flagVar) { “you have reached your goal sale!” }

Integer Flags: 0 is false, any non-zero value is true. Works same as Boolean

Input Validation:

* The process of inspecting data given to a program by the user & determining if it is valid.
* A good program should give clear instruction as to the kind of input that is acceptable, while also assuming that the user has not followed those instructions.

Comparing Characters & Strings:

* Relational operators can also be used to compare character & string objects.

The Conditional Operator:

* Shorthand for if/else statements
* expression ? expression : expression
  + 1st: expression to be tested
  + 2nd: executes if true
  + 3rd: executes if false
* Takes 3 operands – TERNARY

Switch Statement:

* Lets the value of a variable or an expression determine where the program will branch
* Similar to if/else statement, but it tests the value of an integer expression and then uses that value to determine to which set of statements to branch
* An optional default section comes after all the case statements. The program branches to this section if none of the case expressions match the switch expression.
* Selects the execution of the statement often based on a keyboard command VS.
* Allows only the testing of a single expressing against a list of discrete values (less flexible that if/else)

Blocks & Variable Scope:

* The scope of a variable is limited to the block in which it’s defined.

Counter:

Accumulator:

* Keeps track of the sum of numbers that accumulates with each iteration of the loop

**Chapter 5 – Loops & Files**:

**++** and **–** operators:

* Increment or Decrement operators
* Prefix & Postfix ?test? (order)
* Post: evaluates the variable, & then adds or subtracts
* Pre: adds or subtracts, & then evaluates

While loop:

* Executes statements while the condition is true
* Pretest loop – tests its expressions before each iteration
* Good for input validation – repeats while predicted bad data is entered, until acceptable data is entered.
* Good for reading lists of data terminated by a sentinel value.

Do-while loop:

* Always executes action at least once
* Posttest loop – executes its expression after each iteration
* If condition evaluated is true, action continues while it remains true
* User controlled loop? Programmer has no way of knowing the number of times the loop would iterate
* Good for when you always want the loop to iterate at least once.
* Good choice for a repeating menu

\*While & For While loops – conditional loops

For Loop:

* Ideal for performing a known number of iterations
* Pretest loop
* Count-controlled loop
* Steps: initialize counter variable to a starting value, test against max/min value, ++/--
* Can use multiple statements in the initialization and update statements

Break statement:

* Terminates the loop early & program jumps to next line after the loop
* Can be used to give the user the option to terminate the loop by entering in a value.

Continue statement:

* Causes a loop to stop its current iteration and begin at the next iteration.
* All other statements in the body of the loop after the continue will be ignored.
* While loop – program jumps to test expression at top of loop
* Do While loop – program jumps to test expressions at bottom of loop
* For loop – program jumps to update expression, and then test expression.

Infinite loop:

* When the conditional statement/Counter is absent

Sentinel:

* A special value that marks the end of a list of values.
* Often used by asking a user to enter a sentinel at the end of a list of values.
* Cannot be mistaken for one of the values & signals that there are no more values to be entered.
* When a user enters a sentinel, the loop terminates.
* EX: prompt user to enter list of points their team has earned this season. Enter -1 when finished.

GIGO:

* Garbage in garbage out
* Input Validation – checks for integrity/reasonableness
  + Display error & provide more specific instruction