Notes about the class & labs

Program 3.4.4: There is an error in the starting solution, in the main() function. The variable to should be till.

Program 3.4.5: should include using namespace std; in starter code.

Slide 3.10.4: error in example declaration int count = new int; It should be int\* count = new count;

Lab 4.1.1: in the example input, the last line of input is 3 4 4 5. However, in the corresponding example output the Course 3 output shows individual grades listed as 4 3 5. They should be 4 4 5.

Lab 4.2.1: for the example input of 100000, the example output is incorrect. It should be 5000050000, not 50005000.

File cpa\_lab\_4\_2\_7(1)-A.pdf: should be named Lab 4.2.2 Comparing floating-point numbers, not Lab 4.2.1, as there already is one with that number: Range of Integer Types. Also, the starter code is missing a “;” at the end of the last cout instruction.

File cpa\_lab\_4\_5\_6\_(10-B.pdf: starter code does not have a return 0; statement at the end.

File cpa\_lab\_4\_5\_7\_(3)-B.pdf: starter code has an extra ‘}’ at its end.

File cpa\_lab\_4\_5\_7\_(5)-B.pdf: the starter code erroneously suggests that the output should be the sentence that was input. It should be all the matching substrings. Also, the code has an extra ‘}’ at its end.

File cpa\_lab\_4\_5\_7\_(6)-B.pdf: the starter code erroneously suggests that the output should be the sentence that was input. It should be all the matching substrings. Also, the code has an extra ‘}’ at its end.

File cpa\_lab\_4\_5\_7\_(7)-B.pdf: Given the first example input, the implication is that the first requirement be that the password be at least 8 characters long, not exactly 8 characters long.

File cpa\_lab\_4\_5\_7\_(8)-B.pdf: the starter code erroneously suggests that the output should be the template that was input. It should be the processed template. Also, the code has an extra ‘}’ at its end. Also, “template” is a C++ keyword. A different variable name must be used.

File cpa\_lab\_5\_1\_9\_(1)-A.pdf: error on line outputting person.name: should not have () after name. Also missing a “<<” in the function print, before person->age.

File cpa\_lab\_5\_1\_9\_(2)-A.pdf: error in starter code for print function: missing “<<” before area.

File cpa\_lab\_5\_1\_9\_(3)-B.pdf: error in starter code: constructor for LazySquare class has name AdHocSquare.

File cpa\_lab\_5\_3\_10\_(2)-B.pdf: error in starter code: in main(), call to std::cin.getline(command); generates an error. Should be getline(cin,command); The preceding line is missing a “;”, also. The implementation of bool FlightBooking::canceReservations(int number\_ob\_seats); erroneously ends in a semicolon.

File cpa\_lab\_5\_3\_10\_(3)-C.pdf: in starter code, the declaration for FlightBooking::getId() has a misspelled keyword: “retutn”. It should be “return”. Also, the “;” should go inside the braces, not outside of them.

File cpa\_lab\_5\_3\_10\_(5)-C.pdf: error in starter code: in main(), call to std::cin.getline(input); generates an error. Should be getline(cin,input);. Also the last line of output is wrong on several syntax points.

File cpa\_lab\_5\_3\_10\_(6)-B.pdf: error in starter code: method Fraction::times does not state a return type and does not have an ending “}”. The method also does not calculate the product of fractions correctly. Finally, the last example output of 2 11/12 has the wrong sign. It should be   
-2 11/12.

File cpa\_lab\_5\_3\_10\_(9)-B.pdf: error in last example output, should be -1.66667x – 0.66667.

File cpa\_lab\_5\_3\_13\_(3)-A.pdf: error in starter code: in the main function the value popped from the list should not be the loop index i. Change it to value. Also, the starter code pops four values, not three.

File cpa\_lab\_5\_3\_13\_(6)-B.pdf: Starter code for printList has errors.

File cpa\_lab\_5\_3\_13\_(6)-B.pdf: Example output is incorrect.

File cpa\_lab\_6\_4\_4\_(1)-B.pdf: In the starter code, the argument of isValid in the function printValid is incorrect.

File cpa\_lab\_6\_4\_4\_(2)-C.pdf: In the starter code, the argument of isValid in the function printValid is incorrect. There are typos in the starter code: endl is followed by :, not ;.

String functions:

For reference: http://www.cplusplus.com/reference/string/string/

* s.append(t): appends t on the end of s
* s.append(t,m,n): appends the substring of t, starting at position m, of length n, to the end of s.
* s.append(n,ch): appends n copies of ch to the end of s.
* s.assign(n,ch): set s equal to n copies of character ch.
* s.capacity(): amount of storage currently allocated for s.
* s.compare(t): returns 0 if s == t, returns n > 0 if s > t; n < 0 if s < t
* s.compare(m,n,t): let s’ be the substring of s, starting at position m, having n characters. Returns 0 if s’ == t, x < 0 if s’ < t, x > 0 if s’ > t.
* s.compare(m,n,t,p,q): let s’ be the substring of s, starting at position m, having n characters. Let t’ be the substring of t, starting at position p, having q characters. Returns 0 if s’ == t’, x < 0 if s’ < t’, x > 0 if s’ > t’.
* s.empty(): returns 1 if s has no characters, returns 0 otherwise.
* s.erase(m,n): remove a substring from s, starting at position m and consisting of n characters.
* s.find(t) returns position of string t in s, starting search at beginning of s.
* s.find(t, m) returns position of string t in s, starting search at location m in s.
* s.insert(m, t): inserts string t into s starting at position m.
* s.insert(m,n,ch): insert n copies of character ch at position m in s.
* s.insert(m,t,p,q): insert substring of t, starting at position p and having q characters, at position m in s.
* s.length() returns number of characters in s
* s.max\_size(): maximum number of character possible for s.
* s.push\_back(ch): appends the character ch to the end of s.
* s.replace(m,n,t,p,q): replace the section of s, starting at position m and consisting of n characters with the substring of t, starting at position p and consisting of q characters.
* s.reserve(m): allocate m bytes for s.
* s.resize(m,ch): change the size of s to m characters and fill any extra new characters with ch. ch is ‘\0’ by default.
* s.size() returns number of characters in s
* s.substr() returns s
* s.substr(m) returns substring of s, starting at position m
* s.substr(m,n) returns substring of s, starting at position m, having n characters
* s.swap(t): exchange the content of s with that of t.
* string::npos value returned from s.find(t) when t is not found in s.

Constants

* A pointer (the address of some data) is constant, if declared as:  
  int arr[5] = {1,2,4,8,16};  
  int \* const iptr = arr + 2; // the value of \*(arr+2) can change, the value of iptr cannot.  
  char \* const cptr = “Why?”; // The string literal can change, the value of cptr cannot.  
  The point here is that the key word const follows the \*.
* A pointer to a constant is declared as:  
  int arr[5] = {1,2,4,8,16};  
  const int \* iptr = arr + 2;  
  const char \* cptr = “Why?”  
  The data which is pointed to now cannot change, although the pointer can change.  
  The point here is that the key word const precedes the \*.  
  The key word const and the data type (e.g. int or char) can be in either order:  
  const int or int const; either way has the same effect.
* A constant pointer to a constant datum is defined with two instances of the keyword const:  
  int arr[5] = {1,2,4,8,16};  
  const int \* const iptr = arr + 2;  
  const char \* const cptr = “Why?”  
  Here, neither the pointer nor the data can be changed.
* A function parameter that is passed by value can include the const keyword, which has the effect of preventing the parameter from be altered inside the function:  
  int f(const int n){ // n is prevented from being altered in here. }
* A function parameter that is passed by reference can include the const keyword, which has the effect of preventing the parameter from be altered inside the function:  
  int f(const int &n){ // n is prevented from being altered in here. } This prevents any modification of n from propagating back out of the function.
* The value returned from a function can be constrained to be unalterable by using the keyword const in the return type. This is primarily useful with pointers to data that must not be changed:  
  const char \* f(){return “abc”;}  
  String literals are stored in read-only memory and cannot be changed.
* The keyword const is used in the declaration of a field in a class, to cause that field to be unalterable during the lifetime of the object. Such fields must be initialized in the initialization list in all of the class constructors.
* If the keyword const is used at the instatiation of an object, then that objects fields cannot be modified and its methods cannot be invoked: const Class object;
* If a class’s member function is declared as const, then it will not modify the state of the object derived from that class. The keyword const is placed after the parameter list of the function: int get(void) const { return x;}

**Exceptions**

* To use predefined standard exceptions, must have #include <stdexcept>.
* class exception: base for all other predefined exceptions.  
  + class logic\_error: represents all exceptions caused by violation of the logic of the program. These are preventable and statically predictable.  
    - class domain\_error: represents all exceptions caused by data exceeding permissible range.
    - class length\_error: represents all exceptions caused by using illegal values to specify size or length of data aggregates.
    - class out\_of\_range: represents all exceptions caused by using illegal index values or keys while accessing indexed or keyed data collections.
  + class runtime\_error: represents all exceptions caused by circumstances that occur while the program is running. These are unpreventable and are not statically predictable (e.g. network failure).  
    - class range\_error: represents exceptions caused by obtaining computational results that exceed the permissible range.
    - class overflow\_error: represents exceptions caused by obtaining results that are too large to represent any useful value.
    - class underflow\_error: represents exceptions caused by obtaining results that are too small to represent any useful value.
  + class bad\_alloc: represents exceptions arising from invoking new or new [] when the memory requirements cannot be fulfilled.
  + class bad\_exception: this is thrown whenever a function tries to throw an exception of a type that is not specified by its throw specification.