COMP3512 – Lab Exercise 1 (Sep 15–18, 2015)

This is a practical lab exercise that you need to do on a computer. Maximum score: 10

For this exercise, you'll need to write a couple of C++ programs to perform input & output. You must use C++ I/O streams for all input & output. *Note that the 2 programs should be tested using I/O redirection.* In the following, we use ␣ to denote the space character when necessary.

1. Write a C++ program that reads integers from standard input & prints the *non-negative* ones to standard output in octal, decimal & hexadecimal as shown in the following sample output:

```
␣␣␣␣␣␣␣dec␣␣␣␣␣␣␣␣␣oct␣␣␣␣␣␣hex
----------␣-----------␣--------
␣␣␣␣␣␣␣␣23␣␣␣␣␣␣␣␣␣␣27␣␣␣␣␣␣␣17
␣␣␣␣␣␣␣678␣␣␣␣␣␣␣␣1246␣␣␣␣␣␣2A6
␣␣␣␣␣␣␣786␣␣␣␣␣␣␣␣1422␣␣␣␣␣␣312
␣␣␣␣␣␣7777␣␣␣␣␣␣␣17141␣␣␣␣␣1E61
```

   Input is read integer by integer until end of file. Any invalid input (as determined by the input operator) is skipped. For example, the above output could have come from the following input:

```
23
  678  786  abc
   7777
```

   Note that abc is skipped by the program.

2. Write a C++ program that reads floating-point numbers from standard input & prints them & their average to standard output in the format shown by the following sample output:

```
␣␣␣␣␣+␣␣␣␣1234.57
␣␣␣␣␣-␣␣␣␣␣567.25
␣␣␣␣␣+␣␣␣␣␣666.60
␣␣␣␣␣-␣␣␣␣␣␣77.00
-----------------
avg:␣+␣␣␣␣␣314.23
```

   Input is similar to the previous program except that we read floating-point numbers rather than integers. The input for the above could be:

```
1234.567
 -567.246  +666.6
-77
```

COMP3512 – Lab Exercise 2 (Sep 21–25, 2015)

This is a practical lab exercise that you need to do on a computer. Maximum score: 10

In this exercise, you are asked to implement two C++ functions to get user input & use them to implement a simple test program.

The 2 functions to get user input are:

```
bool get_valid_word(const string& prompt, string& word, bool (*is_valid)(const string&));
bool get_valid_int(const string& prompt, int& n, bool (*is_valid)(int));
```

Note that the third parameter to both functions is a function pointer that is used to validate the user input.

Essentially the 2 functions keep on prompting & getting user input until either valid data is obtained or until the user presses the end-of-file key (at the beginning of a line). In the first case, the function returns true; in the latter, it returns false.

Both functions prompt the user to enter data by displaying the string `prompt` (to standard output) & then wait for user input (from standard input). If the user presses the end-of-file key, the function clears the input stream & returns false. Otherwise, a line of input is read & only the first word (if there is one) of the line is used; extra words are ignored. If there are no words in the line, the function simply prompts for input again.

If there is a word,

- for `get_valid_word`, the `is_valid` function is then used to validate the first word — the word is regarded as valid if & only if `is_valid` returns true for it. If the word is valid, it is stored in `word` & the function returns `true`; otherwise, the function prompts for input again.
- for `get_valid_int`, the first word is converted to an integer. If this succeeds, the integer is then validated by the `is_valid` function. If the integer is valid, it is stored in `n` & the function returns `true`. Otherwise (i.e., if either the word cannot be converted to an integer or the integer is invalid), the function prompts for input again.

Use the 2 functions to implement a program that gets user input for student records & writes them to a file using I/O redirection.

A student record consists of an ID (a string) & a score (an integer). The ID must start with the letter 'a' followed by 8 digits & the score must be an integer between 0 & 100 inclusive. (Use the above 2 functions to prompt & get these from the user.) Every time the program gets a valid record (i.e., a valid ID & a valid score), it is written to the file (via standard error) as text. The following sample shows the file format:

```
a12345678␣100␣a23456789␣␣72␣a34567890␣␣␣5␣
```

Note that each record takes up the same number of characters.

Input is terminated by the end-of-file key; when this happens, any partially-entered record (e.g., the user may have already entered the ID but pressed the end-of-file key when prompted for the score) is discarded.

Note that you need to write functions to validate the ID & the score.

COMP3512 – Lab Exercise 3 (Sep 28–Oct 2, 2015)

This is an exercise that you need to do on a computer. Maximum score: 10

In this exercise, you asked to implement a program in C++ to handle simple student records.

A student record consists of an ID & a score. As in the previous lab, the ID consists of an 'a' followed by 8 digits, the score is an integer between 0 & 100 inclusive & records are saved in a file in the same format. For example,

```
a12345678␣100␣a23456789␣␣72␣a34567890␣␣␣5␣
```

Note that each record occupies the same number of characters.

The program takes the name of a record file as a command-line argument. The program opens this file for both reading & writing (in binary mode) when it starts. If the file exists already, its content is truncated. The program then displays the following menu:

```
1. Input records
2. Display records
```

& prompts the user for a choice. The user can exit the program at this menu by pressing the "end-of-file" key.

Inputting records is similar to what you did in the last lab except that when a record (an ID & a score) is successfully entered, it is written to the record file. As before, input is terminated by the end-of-file key but the program goes back to the above menu rather than exiting.

For displaying records, the program repeatedly prompts the user to specify the records to display & displays them in a suitable format. We regard records as numbered, starting from 1. This means that the first record saved in the file has record number 1. The user specifies the records to display by entering an integer:

- If the user enters a positive integer, that integer is the record number of the record to display.
- If the user enters a negative integer, then its absolute value specifies the first record to display & all subsequent records are displayed as well.
- If the user enters 0 or the "end-of-file" key, the program goes back to the above menu.

For example, a user input of 3 asks the program to display the third record (record number 3) if it exists, whereas an input of -3 asks the program to display all records starting from the third record. If the specified records don't exist, an error message is printed.

As in the previous lab, all user input is read a line at a time & only the first word of each line is used. If the input is invalid, it is silently ignored. (Use `get_valid_int` & `get_valid_word` from the previous lab.)

Note: You must use some form of seeking to locate the records.

COMP3512 – Lab Exercise 7 (Nov 9–Nov 13, 2015)

This is a practical lab exercise that you need to do on your computer. Maximum score: 10

Consider the following set of 7-digit primes:

4556557, 5456557, 5545567, 5554567, 6455557, 6554557, 6555457, 6555547

It is easy to see that the numbers are permutations of one another. The set consists of 8 elements.

Here is another set of 7-digit primes that are permutations of one another:

2040427, 2044027, 2204047, 4002247, 4004227, 4022407, 4040227, 4200247, 4202047, 4204027, 4240207

This set has 11 elements.

Write a C++ program to find the largest set of 7-digit primes that are permutations of one another. What is the size of the set & what are the primes?

COMP3512 – Lab Exercise 8 (November, 2015)

This is a lab exercise that you need to do on a computer. Maximum score: 10

For this exercise, you are basically asked to implement a `Student` class & an `OptionStudent` class (for students in an option) publicly derived from the `Student` class.

A student basically has an ID & a name. A name consists of a first name & a last name. You may want to put this in a separate `Name` class.

```
class Student {
public:
  // provide ctor(s)
  virtual ~Student() {}
  virtual void print(std::ostream& os) const;    // see sample output
  virtual void display(std::ostream& os) const;  // see sample output
  // friend operator>> (used to read a student "saved" by the print method)
private:
  std::string  id_;          // e.g. a11111111
  Name         name_;        // e.g. homer simpson
  static bool isValidId(const std::string& id);  // check for valid ID
  // additional helper functions if necessary
};
```

A valid ID must start with the letter 'a', followed by 8 digits. The first & last names are each one word (with no leading or trailing whitespace). Internally, the letters in the name are stored in lowercase. For simplicity, put all your implementation in the header files.

An option student has an option & the term he/she is in. We also use a static counter to keep track of the number of option students:

```
class OptionStudent: public Student {
public:
  // provide ctors & dtor
  static size_t getCount() { return count_; }
  virtual void print(std::ostream& os) const override;    // see sample output
  virtual void display(std::ostream& os) const override;  // see sample output
  // friend operator>> (used to read an option student "saved" by the print method)
private:
  int           term_;    // 3, 4, ...
  std::string   option_;  // 1 word, e.g., IS, CS, ...
  static size_t count_;   // keep track of the number of option students in existence
};
// implement operator<< by calling the print method
```

Put your implementation in a cpp file. Be sure to define `OptionStudent::count_`.

The print & display methods have different output. For example,

```
Student        s1("a12345678", "Homer", "Simpson");
OptionStudent  s2("a23456789", "Bart", "simpson", 3, "IS");
s1.print(cout);
cout << "\n*****\n";
s1.display(cout);
cout << "*****\n";
s2.print(cout);
cout << "\n*****\n";
s2.display(cout);
// Student  s2("AAA", "carl", "carlson");  // should throw exception
```

should output

COMP3512 – Review Exercise 1 (Feb, 2015)

1. Write a C++ program that uses C++-style I/O to read floating-point numbers from a text file & print them & their average to standard output. The name of the input file is "numbers.txt". The file is processed line by line. If the first word in a line starts with a number, that number is used. Otherwise, the line is skipped. Reading continues until end-of-file is reached. The following shows possible file content:

   ```
   123.5
   1.2abc   # should read in 1.2
   hello    # invalid; should be skipped
   2.3  4.5 # should read in 2.3 only
   ```

   The program prints each number it reads, finally followed by the average. All numbers are printed to 2 decimal places on separate lines & with a minimum width of 10. For example, 123.5 would be printed as ␣␣␣␣123.50 (Note: ␣ is the space character.)

2. Consider the following containers:

   ```
   vector<int>  v{3,2,7,6,8,1,4,5};
   list<int>    lst{9,8,7,6}
   ```

   Note: Parts (a)–(d) of this question use the v & lst containing the values specified above.

   (a) Use the STL copy algorithm to display the last 5 integers in the vector v to cout, with each integer on a separate line.

   (b) Use the STL copy algorithm to copy all the integers in lst to v, overwriting the integers 7, 6, 8, 1 originally in v. (Note: After the operation, v should contain 3, 2, 9, 8, 7, 6, 4, 5.)

   (c) Use the STL transform algorithm to append to the vector v numbers calculated by tripling each integer in lst. (Note: After the operation, v should contain 3, 2, 7, 6, 8, 1, 4, 5, 27, 24, 21, 18.)

   (d) Use the STL sort algorithm to sort the numbers in v in descending order. Do this in 2 ways: using a function object & using a lambda expression. (Note: After this operation, v should contain 8, 7, 6, 5, , 4, 3, 2, 1.)

   (e) Suppose more numbers have been added to the original vector v. Use the STL for_each algorithm together with an appropriate function object to obtain 2 counts: the number of integers in v that are less than 0 & the number of integers in v that are greater than 100. Print the 2 counts to cout.

3. Without calling other generic algorithms, implement the copy_if algorithm that was introduced into C++-11. Its "prototype" is

   ```
   template<typename InputIterator, typename OutputIterator, typename Predicate>
   OutputIterator
   copy_if(InputIterator first, InputIterator last, OutputIterator result, Predicate f);
   ```

   copy_if copies those elements in the half-open range [first, last) that satisfy f to the destination specified by result & returns an iterator just past the last copied element in the destination.

   For example, if v is a vector containing 3,2,7,6,8, lst is a list containing 10,11,12,13,14,15 & is_even is defined by

   ```
   bool is_even(int n) { n % 2 == 0; }
   ```

   then after executing

   ```
   list<int>::iterator it = copy_if(v.begin(), v.end(), lst.begin(), is_even);
   ```

   lst would contain 2,6,8,13,14,15 & it would point to 13.