

Ch03-StdInputOutput

August 8, 2020

1 3 Standard Input and Output

1.1 Topics

- common way to input and output
- printing variables and values onto monitor or console
- reading data from keyboard
- composing programs

1.2 3.1 Input and output (IO)

- IO operations are fundamental to computer programs
- C++ IO occurs in streams (sequence of bytes)
- programs must be able to read data from varieties of input devices (input operation)
 - streams of bytes flow from keyboard, disk drive, network connection, etc. to main memory, RAM (Random Access Memory)
- programs must be able to write data to varieties of output devices (output operation)
 - stream of bytes flow from RAM to monitor, disk drive, network connection, etc.
- this chapter covers standard input and output
- reading from and writing to disk drive or files is covered in File IO chapter

1.3 3.2 Standard output stream

- a program may need to display data or results of computation to users
- a common way to display results is by printing them to common output (monitor)
 - also called console
- we've printed hello world string to console in Chapter 1
- similarly, we can print literal values or data stored in variables
- use **cout** statement defined in `<iostream>` library and **std** namespace
- output statement syntax:

```
cout << varName1 << varName2 << "literal values";
```

- `<<` called **stream insertion operator** inserts values to output stream
- multiple values are separated by `<<` operator
- **endl** operator adds a new line

```
[16]: // include required library
      #include <iostream> // cout
```

```
// use required namespace
using namespace std; //std namespace defines cout, endl, etc.
```

```
[3]: cout << "Hello World!" << endl;
      cout << 100 << 2.5f << ' ' << 3.99 << 'A' << "some text as string";
      cout << "continue printing stuff in next line...?" << endl;
```

```
Hello World!
1002.5 3.99A some text as stringcontinue printing stuff in next line...?
```

```
[4]: // declaring and printing variables
      #include <string>

      string name = "John Doe";
      char MI = 'A';
      int age = 25;
```

```
[5]: // outputting variables
      cout << "name = " << name << endl;
      cout << "MI = " << MI << " and age = " << age << endl;
```

```
name = John Doe
MI = A and age = 25
```

```
[6]: bool done = false;
      float temperature = 73;
      float richest_persons_networth = 120000000000; // 120 billion
      float interestRate = 4.5;
      float length = 10.5;
      float width = 99.99f; // can end with f for representing floating point number
      double space_shuttle_velocity = 950.1234567891234567 // 16 decimal points
```

```
[7]: // cout can continue in multilines
      cout << "temperature: " << temperature << " age: " << age
          << " richest person's worth: "
          << richest_persons_networth << endl;
      cout << "interest rate: " << interestRate << endl;
      cout << "length: " << length << " and width = " << width << endl;
      cout << "space_shuttle_velocity: " << 950.1234567891234567 << endl;
```

```
temperature: 73 age: 25 richest person's worth: 1.2e+11
interest rate: 4.5
length: 10.5 and width = 99.99
space_shuttle_velocity: 950.123
```

```
[8]: // outputting string variables
      cout << "Hello there, " << name << '!' << endl;
```

Hello there, John Doe!

```
[9]: // more string variables
string address1 = "1100 North Ave";
string state_code = "CO";
string country = "USA";
```

```
[10]: cout << "CMU's address:\n"
      << address1 << endl
      << "Grand Junction, " << state_code << ' ' << 81501 << endl
      << country << endl;
```

```
CMU's address:
1100 North Ave
Grand Junction, CO 81501
USA
```

1.3.1 Escape sequences

- some letters or sequence of letters have special meaning to C++
- pair of single quote is used to represent a character type
- pair of double quotes is used to represent a string type
- how can we store single or double quotes as part of data?
 - e.g., we need to print: **“Oh no!”**, **Alice exclaimed**, **“Bob’s bike is broken!”**
 - use backslash \ (escape character) to escape the special meaning
- characters represented using escape character are called escape sequences
 - \n - new line
 - \\ - back slash
 - \t - tab
 - \r - carriage return
 - \' - single quote
 - \" - double quote

```
[11]: cout << "What's up\n Shaq\tO'Neal?";
```

```
What's up
Shaq   O'Neal?
```

```
[12]: char quote = '\\';
```

```
[13]: quote
```

```
[13]: '\\'
```

```
[14]: cout << "\"Oh no!\", Alice exclaimed, \"Bob's bike is broken!\"";
```

```
"Oh no!", Alice exclaimed, "Bob's bike is broken!"
```

```
[15]: cout << "how many back slashes will be printed? \\\\";
```

how many back slashes will be printed? \\

1.4 3.3 Standard input stream

- often, data must be read from standard input stream or keyboard
 - e.g. most interactive programs
- must include `<iostream>` library for standard input
- must use `std` namespace
- use `cin >>` statement
- syntax:

```
cin >> var1 >> var2 >> ...;
```

- `>>` stream extraction operator
 - extracts one or more data from input stream
- must always use variables of appropriate types
- while scanning input stream, `>>` ignores leading whitespaces and stops at a trailing whitespace
- let's say we have a stream of data separated by a whitespace: 10 11 15.5 A
 - we can parse and extract as following
 - * `cin >> num1 >> num2 >> num3 >> alpha;`
 - given num1 and num2 are int, num3 is float or double and alpha is char

1.4.1 inputting numerical data

- must store the extracted input data into appropriate numerical variables
- `>> int variables` extracts whole numbers from input stream; stops at anything else
- `>> float or double variables` extracts numbers including decimal points; stops at anything else

```
[17]: // include required libraries
#include <iostream> //cin, cout

using namespace std;
```

```
[15]: int num1;
// prompt user to enter a whole number
cout << "enter a whole number: ";
cin >> num1;
cout << "You entered: " << num1 << endl;
```

enter a whole number: 10
You entered: 10

```
[10]: // can extract multiple integers
int num2;
cout << "enter two whole numbers separated by space: ";
```

```
cin >> num1 >> num2;
cout << num1 << '+' << num2 << '=' << num1+num2 << endl;
```

enter two whole numbers separated by space: 10 20
10+20=30

```
[11]: // extracting int and float
float num3;
cout << "enter a whole number and a floating point number separated by space: ";
cin >> num1 >> num3;
cout << num1 << " + " << num3 << " = " << num1+num3 << endl;
```

enter a whole number and a floating point number separated by space: 5 9.9
5 + 9.9 = 14.9

```
[12]: // let's enter 10 11 15.5 A and store them into corresponding variables
int n1, n2;
float n3;
char alpha;
```

```
[13]: // let's not prompt; but simply enter 10 11 15.5 A
cin >> n1 >> n2 >> n3 >> alpha;
```

10 11 15.5 A

```
[14]: // let's echo the entered values
cout << n1 << " " << n2 << " " << n3 << " " << alpha;
```

10 11 15.5 A

1.4.2 input failure

- if input data and variable type mismatched, cin will enter into a fail state
 - won't be able to extract data anymore
- Note: Jupyter notebook may crash or simply not work as expected when input fails

```
[15]: // variable to store whole number
int number;
```

```
[16]: cout << "Enter a number: ";
cin >> number;
cout << "You entered " << number;
// try entering whole number; whole number and characters, characters and
↪ number, etc.
```

Enter a number: adf
You entered 0

[16]: @0x107733ec0

1.4.3 inputting string data

- two ways depending on the string data type
- string without whitespace or single word can be extracted using `>>` stream extraction operator
- string with whitespace must be extracted using **getline** function
- syntax:

```
getline(cin, strVar);
```

- getline reads the entire line including whitespaces and the **newline** at the end
 - newline is read but ignored and not stored as a part of string

```
[17]: string player_name;
```

```
[18]: cout << "Enter your first name: ";  
cin >> player_name;  
cout << "Hello there, " << player_name << endl;  
// run it with just firstname and then with fullname; notice the value of  
↪player_name
```

```
Enter your first name:  
John Smith  
Hello there, John
```

[18]: @0x107733ec0

```
[19]: // string with spaces  
cout << "Enter your full name: ";  
getline(cin, player_name);  
cout << "Hello there, " << player_name << endl;
```

```
Enter your full name: John Smith  
Hello there, John Smith
```

1.4.4 Note

- `getline()` reads and discards newline character (`\n`)
- `>>` stops before the trailing newline character leaving it in the input stream
- must explicitly read and discard newline character if `getline` is used subsequently
- use **ws** whitespace manipulator
 - `ws` operator extracts as many whitespace characters as possible from the current position in the input stream
 - extraction stops as soon as a non-whitespace character is found
 - e.g. `cin >> number >> ws;`
 - reads and discards whitespace(s) including `\n` after number value in input stream

1.4.5 demo program

- program that demonstrates the above caveat is found here [demo_programs/Ch03/stdio.cpp](#)

1.5 3.4 Composition

- similar to composing an essay or music
 - start with basic elements and combine them to build something more bigger and meaningful work
- we use the same basic principle of **composition** in coding
 - take small building blocks
 - * variables, values, expressions(operators), statements (input, output), etc.
 - compose something meaningful or solve a problem

1.5.1 example 1: find area and perimeter of a rectangle

- algorithm steps:
 1. get values for length and width of a rectangle
 - calculate area and perimeter using the following equations
 - * $\text{area} = \text{length} \times \text{width}$
 - * $\text{perimeter} = 2 \times (\text{length} + \text{width})$
 - display the results

```
[20]: // ex.1 program
      // variables to store length and width
      float rect_length, rect_width;
```

```
[21]: // 1 get values;
      // a. can be hardcoded literal values
      rect_length = 10.5; //hardcoded
      rect_width = 5.5;
```

```
[22]: // b. or can be read from keyboard
      cout << "Enter length and width of a rectangle separated by space: ";
      cin >> rect_length >> rect_width;
```

Enter length and width of a rectangle separated by space: 10.5 5.5

```
[23]: // 2 and 3: calculate and display the area and perimeter
      cout << "area of the rectangle: " << rect_length * rect_width << endl;
      cout << "perimeter of the rectangle: " << 2*(rect_length+rect_width) << endl;
```

area of the rectangle: 57.75

perimeter of the rectangle: 32

1.5.2 demo programs

- see complete program here [demo_programs/Ch03/composition.cpp](#) or at <https://repl.it/@rambasnet/CS1-Rectangle>

1.5.3 example 2: convert decimal to binary

- let's convert $(13)_{10}$ to binary $(?)_2$?
 - from manual calculation we know $(13)_{10} \rightarrow (1101)_2$
- let's use algorithm defined in chapter 2:
 1. repeatedly divide the decimal number by base 2 until the quotient becomes 0
 2. collect the remainders in reverse order
 - the first remainder is the last (least significant) digit in binary
- let's try to convert the above algorithm into C++ code

```
[1]: #include <iostream> // cin, cout
#include <string> // string, to_string

using namespace std; // std::cin, std::cout, std::endl, etc.
```

```
[2]: // decimal to binary conversion requires to calculate both quotient and
    ↪ remainder
const int divisor = 2; // divisor is constant name whose value can't be changed
    ↪ once initialized with
int dividend;
int quotient, remain;
string answer; // collect remainders by prepending as a string
```

```
[4]: dividend = 13;
answer = "";
```

```
[5]: remain = dividend%divisor;
quotient = dividend/divisor;
cout << dividend << '/' << divisor << " quotient: " << quotient << " remainder:
    ↪ " << remain << endl;
answer = to_string(remain) + answer; // prepend remainder to answer
// is quotient 0?
```

13/2 quotient: 6 remainder: 1

```
[5]: "1"
```

```
[6]: // further divide quotient
remain = quotient%divisor;
quotient = quotient/divisor;
cout << dividend << '/' << divisor << " quotient: " << quotient << " remainder:
    ↪ " << remain << endl;
answer = to_string(remain) + answer;
// is quotient 0?
```

13/2 quotient: 3 remainder: 0

```
[6]: "01"
```



```
[7]: // further divide quotient
      remain = quotient%divisor;
      quotient = quotient/divisor;
      cout << dividend << '/' << divisor << " quotient: " << quotient << " remainder:␣
      ↪" << remain << endl;
      answer = to_string(remain) + answer;
      // is quotient 0?
```

13/2 quotient: 1 remainder: 1

[7]: "101"

```
[8]: // further divide quotient
      remain = quotient%divisor;
      quotient = quotient/divisor;
      cout << dividend << '/' << divisor << " quotient: " << quotient << " remainder:␣
      ↪" << remain << endl;
      answer = to_string(remain) + answer;
      // is quotient 0?
```

13/2 quotient: 0 remainder: 1

[8]: "1101"

```
[10]: // no more division; display the answer
      cout << dividend << " base 10 = " << " binary " << answer << endl;
```

13 base 10 = binary 1101

1.5.4 A complete C++ program to convert decimal to binary

- basic building blocks covered so far is able to find the solution in Jupyter notebook
 - however, we've not learned enough to write a complete program that is intuitive and complete yet!
- we'll revisit this problem as we learn more concepts, such as conditional statements and loops

1.6 3.5 Exercises

1. Write a C++ program including algorithm steps that calculates area and perimeter of a circle.
2. Write a C++ program including algorithm steps that calculates Body Mass Index (BMI) of a person.
 - More information on BMI - https://www.nhlbi.nih.gov/health/educational/lose_wt/BMI/bmicalc.htm
 - Formula [here](#).
 - a sample solution is provided here [demo_programs/Ch03/BMI.cpp](#) and here: <https://repl.it/@rambasnet/CS1-BMI>
 - directly run the program at: <https://cs1-bmi.rambasnet.repl.run/>
3. Write a C++ program including algorithm steps that calculates area and perimeter of a triangle given three sides.

- Hint: use Heron's formula to find area with three sides.
4. Write a C++ program that converts hours into seconds.
 - e.g. given 2 hours, program should print 7200 as answer.
 5. Write a C++ program that converts seconds into hours, minutes and seconds.
 - e.g. given 3600 seconds, program should print 1 hour, 0 minute and 0 second.
 - e.g. given 3661 seconds, program should print 1 hour, 1 minute and 1 second.
 - Hint: use series of division and module operators

1.7 3.6 Kattis Problems

1. Solving for Carrots - <https://open.kattis.com/problems/carrots>
 - a simple standard input/output problem; just print the second number in first line
 - see sample solution in [demo_programs/Ch03/carrots] folder
2. R2 - <https://open.kattis.com/problems/r2>
 - simply print: 2*S-R1
3. Spavanac - <https://open.kattis.com/problems/spavanac>
 - convert min+hour to minute; subtract 45 and convert the result back to hour minute and print it

1.8 3.7 Testing Kattis provided samples

- one way to check for the sample input and output is manually typing the input and comparing the results
 - input can be long and output can be tedious to compare
 - Kattis expects output to be 100% accurate to the space
- here's the better way to automate the process:
- download the samples provided in a compressed .zip file
- unzip the file; it'll create a folder with the same name as the problem name or zip file name
- create .cpp solution file inside the same folder where the sample files are
- follow the following instructions:
- open a terminal on Mac/Linux/WSL or cmd prompt on Windows
- change working directory to a problem folder, e.g. carrots

```
$ cd carrots
```

```
$ pwd
```

- compile using g++

```
$ g++ -std=c++14 cold.cpp
```

- run kattis provided sample test cases e.g. if 1.in and 1.ans are sample test files:
- on Mac/Linux/WSL Terminal
- read the sample 1.in and feed it to ./a.out program and feed the answer to diff to compare against 1.ans

```
$ cat 1.in | ./a.out | diff - 1.ans
```

```
$ cat 2.in | ./a.out | diff - 2.ans
```

- on Windows cmd prompt:

```
> type 1.in | a.exe > out1.txt  
> FC out1.txt 1.ans
```

- once your program provides correct result as shown in the output for the corresponding output, now is a good time to upload it to the Kattis to test against all the hidden test samples.

1.9 3.8 Summary

- this chapter covered reading data from common input stream (standard input)
- this chapter covered writing data to common output stream (standard output)
- covered escape character, sequences and their usage
- we also learned about composing more meaningful programs with two examples
- exercises and problems with sample solutions

[]: