StdInputOutput

July 29, 2021

1 Standard Input and Output

1.1 Topics

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

1.2 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 devices, 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 devices, etc.
- this chapter covers how C++ handle standard input and output
- reading from and writing to disk drive or files is covered in File IO chapter

1.3 Standard output stream

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

```
cout << varName1 << varName2 << "literal values" << ' ' << 100 << '\n';
```

- **`<<` stream insertion operator** inserts values to output stream
- multiple values are separated by `<<` operator
- `endl` operator or `\n` escape character end line to continue writing in next line

- the following code demonstrates standard output stream

[1]:
```
// 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 3.99Asome 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++
  - e.g., pair of single quote is used to represent a character data, e.g. 'A' or ' ' (space)
  - and pair of double quotes is used to represent a string type, e.g., "Hello World!"
- 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!"**
  - we use backslash \ (escape character) to escape the special meaning of single and double quotes or other characters
- 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!"
```

[2]:
```
cout << "how many back slashes will be printed? \\\\";
```

```
how many back slashes will be printed? \\
```

1.4 Standard input stream

- often, data must be read from standard input stream or keyboard
  - e.g. most interactive programs with Graphical User Interface (GUI) or Command Line Interface (CLI)
- must include `<iostream>` library for standard input
- must use **std** namespace
- use **cin** » statement
- syntax:

  `cin >> var1 >> var2 >> ...;`
- `>>` stream extraction operator extracts data/value 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 whitespaces: `10 11 15.5 A`
  - we can parse and extract it as following:

  `cin >> num1 >> num2 >> num3 >> alpha;`

  - given num1 and num2 are of type int or long, num3 is float or double and alpha is char

1.4.1 Inputting numerical data

- we must store the extracted numerical 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 not be able to extract the data from the stream
  - `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

- if the Jupyter crashes or stops working, your must restart the Kernel: `Kernel -> Restart`

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

[16]:
```
cout << "Enter a number: ";
cin >> number;
cout << "You entered " << number;
// Play with it:
// try entering an integer then whole number and characters then characters and␣
 ↪number, etc.
```

```
Enter a number: adf
You entered 0
```

[16]:
```
@0x107733ec0
```

1.4.3 Inputting string data

- we can read string data in two ways depending on if the string has a space (phrase) or not (word)
- string without space or single word can be extracted using `>>` stream extraction operator
- a single string data or line with spaces must be extracted using `getline( )` function
- reading syntax:

  ```
  getline(cin, strVar); // reading a line from std input and storing into strVar
  ```

- `getline()` reads the entire line including whitespaces including the `\n` newline
  - newline is extracted from the input stream and discarded

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

[18]:
```
cout << "Enter your first name: ";
cin >> player_name;
cout << "Hello there, " << player_name << endl;
// run it wih 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, discards and stops at newline character (`\n`)
- `>>` stops before the trailing newline character leaving it in the input stream
- must explictly read and discard newline character if getline is used after `>>`
- 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

  `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 demos/stdio/demo1/main.cpp

1.5 Composition

- similar to composing an essay or music
  - start with basic elements and combine them to build something bigger and meaningful work
- we use the same basic principle of **composition** in programming
  - 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
  2. calculate area and perimeter using the following equations
     - area = length x width
     - perimeter = 2 x (length + width)
  3. display the results

[2]:
```
// ex.1 program
// variables to store length and width
float rect_length, rect_width;
```

[3]:
```
// step 1 get length and width values
// a. can be hardcoded literal values
rect_length = 10.5; //hardcoded
rect_width = 5.5;
```

[6]:
```
// step 1.b or can be read from std input
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: 11.2 6.6
```

[7]:
```
cout << "Rectangle's length = " << rect_length << " and width = " << rect_width;
```

```
Rectangle's length = 11.2 and width = 6.6
```

[8]:
```
// step 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: 73.92
perimeter of the rectangle: 35.6
```

1.5.2 demo programs

- see the complete program here demos/stdio/rectangle/main.cpp

1.5.3 example 2: convert decimal number to binary

- let's convert $(13)_{10}$ to binary $(?)_2$?
  - from manual calculation in Chapter 02, we know: $(13)_{10}$ -> $(1101)_2$
- let's use algorithm defined in Chapter 02:
  1. repeteadly divide the decimal number by base 2 until the quotient becomes 0
  2. collect the remainders in reverse order
     - the first remainder becomes the last bit (least significant) in binary
- let's try to convert the above algorithm into C++ code

[1]:
```
#include <iostream> // cin, cout
#include <string> // basic_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 contant name whose value can't be changed␣
↪once initialized
int dividend;
int quotient, remain;
string answer; // collect remainders by prepending as a string
```

[3]:
```
answer = ""; // variable to collect the binary answer
quotient = 13; //start with the decimal 13
```

[4]:
```
// copy the quotient into dividend to divide it
dividend = quotient;
remain = dividend%divisor; // find the remainder
quotient = dividend/divisor; // find the quotient
// print intermediate results; help us see and plan further computation
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
```

[4]:
```
"1"
```

[5]:
```
// further divide quotient
dividend = quotient;
remain = dividend%divisor;
quotient = dividend/divisor;
// print intermediate results; help us see and plan further computation
cout << dividend << '/' << divisor << " => quotient: " << quotient << "␣
 ↪remainder: " << remain << endl;
answer = to_string(remain) + answer; // prepend remainder to answer
// is quotient 0?
```

```
6/2 => quotient: 3 remainder: 0
```

[5]:
```
"01"
```

[6]:
```
// further divide quotient
dividend = quotient;
remain = dividend%divisor;
quotient = dividend/divisor;
// print intermediate results; help us see and plan further computation
cout << dividend << '/' << divisor << " => quotient: " << quotient << "␣
 ↪remainder: " << remain << endl;
answer = to_string(remain) + answer; // prepend remainder to answer
// is quotient 0?
```

```
3/2 => quotient: 1 remainder: 1
```

[6]:
```
"101"
```

[7]:
```
// further divide quotient
dividend = quotient;
remain = dividend%divisor;
quotient = dividend/divisor;
// print intermediate results; help us see and plan further computation
```

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

```
1/2 => quotient: 0 remainder: 1
```

[7]:
```
"1101"
```

[9]:
```
// stop division; display the answer
cout << "13  decimal = " << answer << " binary " << endl;
```

```
13  decimal = 1101 binary
```

1.5.4 Above code as a complete C++ program

- see demos/stdio/decToBin/main.cpp

1.5.5 A generic C++ program to convert any decimal to binary

- basic building blocks covered so far is able to find the solution using Jupyter notebook
  - however, we've not learned enough to write a generic program that can convert any integer into binary, just yet!
- we'll revisit this problem as we learn more concepts, such as conditional statements and loops

1.6 Labs

1. Standard IO Lab
   - write a C++ program that produces the following output on console
   - use the partial solution provided in labs/stdio/main.cpp
   - observe and note how the special symbols such as single quote, double quotes and black slashes
   - run the program as it is using the provided make file in the stdio folder
   - complete the rest of the ASCII Art by fixing all the FIXMEs
   - write #FIXED next to each FIXME

```
  |\_/|       ******************************     (\_/)
 / @ @ \      *      ASCII Art               *    (='.'=)
( > 0 < )     *      Author: <Your Name>     *  ( " )_( " )
  >>x<<       *      CS Foundation Course    *
  / 0 \       ******************************
```

1.7 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 exercises/stdio/BMI/main.cpp

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
6. Convert your full name into binary code using Jupyter Notebook.

1.8 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
   - Hint: simply print P
   - see sample solution in `demos/stdio/carrots` folder
2. R2 - https://open.kattis.com/problems/r2
   - Hint: simply output $2 * S - R1$
3. Spavanac - https://open.kattis.com/problems/spavanac
   - Hint: convert min+hour to minute; subtract 45 and convert the result back to hour and minute and print them
4. Add Two Numbers - https://open.kattis.com/problems/addtwonumbers
   - Hint: read two numbers and print their sum
5. Echo Echo Echo - https://open.kattis.com/problems/echoechoecho
   - Hint: read the word; print the word three times
6. The Last Problem - https://open.kattis.com/problems/thelastproblem
   - Hint: read the name (may have spaces in between) and print the name as stated

1.9 Testing Kattis provided samples

- one way to check for the sample input and output is by 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% accuracte to the space

1.9.1 Recommended way to automate the process to solve Kattis problems

- 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 a `problemName.cpp` solution file inside the same folder where the sample files are
- then do the following steps:
  - open a terminal on Mac/Linux/WSL
  - change working directory to a problem folder, e.g. carrots

    ```
    pwd
    cd <path to carrots folder>
    ls
    ```
  - directly compile using g++ or create and use a Makefile

    ```
    g++ -std=c++17 carrots.cpp
    ```

  - run Kattis provided sample test cases e.g. if 1.in and 1.ans are corresponding sample test files
  - read the sample 1.in and pipe it to `a.out` program and pipe the answer from the program to diff to compare against 1.ans

    ```
    cat 1.in | ./a.out | diff - 1.ans
    cat 2.in | ./a.out | diff - 2.ans
    ```
  - if the program's answer is correct, you'll not see any difference or output on the terminal
- once your program provides correct result as shown in the corresponding output, upload your `.cpp` source file to the Kattis to test against all the hidden test samples
  - Kattis will compile and execute your program to test against the other samples
- Kattis will either accept your solution or reject with some simple feedback such as wrong answer

1.10 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

[ ]: