CS1-Review

August 8, 2020

1 CS1 Review: Basic concepts & C++ fundamentals

- Note: use your installed C++ compiler or online compilers in case C++ kernel crashes on this notebook or doesn't work in some cases:
  - `https://repl.it/`
  - `https://coliru.stacked-crooked.com/`
  - http://cpp.sh

1.1 Table of Contents

– Section **??**

1.2 Fundamental concepts/building blocks

- Data types and variables
- Input/Output
- Math operations
- Decision/Conditionals
- Loops

1.3 Headers and helper functions

- run include headers and helper function cells right below if Kernel crashes or is restarted

[1]:
```
// headers and namespace required in this notebook demo codes
#include <iostream> //cin, cout
#include <cstdio> //printf
#include <cstring> // funciton to work with cstring
#include <string> // string functions
#include <fstream> // file io
#include <iomanip> // output formatting

using namespace std;
```

1.4 fundamental data types

https://en.cppreference.com/w/cpp/language/types - void : type with an empty set of values - bool : true or false - int : integer/whole number - signed int : signed (positive and negative) representation - default - unsigned int : unsigned (only positive) representation - short : target type will have width of atleast 16 bits - long : width of at least 32 bits - long long : width of at least 64 bits - size_t : unsigned int type - int32_t : signed int 32 - int64_t : signed int 64 - char : signed char representation - float : single precision float (32 bit) - double : double precision (64 bit) - long double: extended precision floating point type

- No fundadamental type available to work with string data (array of characters or buffer)

1.5 variables

- identifier or named memory location that allows us to store data
- syntax to declare a variable:

```
type varName;
```

- know the rules of naming identifiers

[2]:
```
// declaring variables
int x, y, z;
string buffer;
float test1, test2, test3;
```

[3]:
```
// variable declaration and initialization
// = assignment operator;
bool a = true;
char b = 'Z';
short c = 100;
int d = -2000000000;
unsigned int dd = 23232; // must be positive value only!
long e = 2000000000;
long long f = 123456789;
size_t g = 111; //same as unsigned long
int64_t h = 2345;
float i = 123.1234567;
double j = 1234.123456789;
long double k = 12112.1212121211121;
string l = "some string";
```

[4]:
```
cout << "sizeof(bool) = " << 8*sizeof(bool) << " bits." << endl;
// printf("sizeof(b) = %lu\n", sizeof(b)*8); doesn't work!
cout << "sizeof(b) = " << 8*sizeof(b) << " bits." << endl;
cout << "sizeof(short) = " << 8*sizeof(short) << " bits." << endl;
cout << "sizeof(int) = " << 8*sizeof(int) << " bits." << endl;
cout << "sizeof(unsigned int) = " << 8*sizeof(unsigned int) << " bits." << endl;
cout << "sizeof(long) = " << 8*sizeof(long) << " bits." << endl;
cout << "sizeof(long long) = " << 8*sizeof(long long) << " bits." << endl;
cout << "sizeof(size_t) = " << 8*sizeof(size_t) << " bits." << endl;
cout << "sizeof(int32_t) = " << 8*sizeof(int32_t) << " bits." << endl;
cout << "sizeof(int64_t) = " << 8*sizeof(int64_t) << " bits." << endl;
cout << "sizeof(float) = " << 8*sizeof(float) << " bits." << endl;
cout << "sizeof(double) = " << 8*sizeof(double) << " bits." << endl;
cout << "sizeof(long double) = " << 8*sizeof(long double) << " bits." << endl;
cout << "sizeof(string) = " << 8*sizeof(string) << " bits." << endl;
```

```
sizeof(bool) = 8 bits.
sizeof(b) = 8 bits.
sizeof(short) = 16 bits.
sizeof(int) = 32 bits.
sizeof(unsigned int) = 32 bits.
sizeof(long) = 64 bits.
sizeof(long long) = 64 bits.
sizeof(size_t) = 64 bits.
sizeof(int32_t) = 32 bits.
sizeof(int64_t) = 64 bits.
sizeof(float) = 32 bits.
sizeof(double) = 64 bits.
sizeof(long double) = 128 bits.
sizeof(string) = 192 bits.
```

1.6 type casting

https://en.cppreference.com/w/cpp/string/basic_string - converting one type into another if possible - stoi(), stol(), stoll() : converts a string to a signed int - stoul(), stoull() : converts a string to unsigned int - stof(), stod(), stold() : converts a string to float - to_string() : converts an int or float to string

[5]:
```
int id = stoi("111");
float PI = stof("3.1416");
string strNum = to_string(1000.99);
```

[6]:
```
int num = stoi("1234");
cout << num+10;
```

```
1244
```

[7]:
```
float price = stof("10.99");
cout << price*10;
```

```
109.9
```

[8]:
```
strNum = to_string(100.99);
cout << strNum + "99" << endl;
```

```
100.99000099
```

1.7 input/output

- standard input/output
  - iostream: cin, cout
    - getline - read the whole line (including \n) into a string
    - \n is read and discarded
  - cstdio: printf, scanf
- file input/output
  - fstream, ifstream, ofstream
  - steps working with files:
    1. declare file handlers
    2. open file
    3. check if file opened successfully
    4. read from or write to file
    5. close file

[9]:
```
int fileio() {
    ifstream fin;
    string line;
    fin.open("README.md");
    if (!fin)
        cout << "file couldn't be opened!" << endl;
    else {
```

```
        while(!fin.eof()) {
            getline(fin, line);
            cout << line << endl;
        }
    }
    return 0;
}
```

[10]:
```
fileio();
```

Data Structures & Applications
- Jupyter Notebooks for teaching and learning Data structures using C++
- Some chapters of notebooks are based on open-source textbook: [CS2 Software Design & Data Structures](https://opendsa-server.cs.vt.edu/ODSA/Books/CS2/html/IntroDSA.html) from Virginia Tech's OpenDSA Project

Requirements
- Linux/MacOS/WSL on Windows (Not tested on Windows itself)
- Jupyter Notebook
- xeus-cling notebook kernel
- git

Install required tools
- Note: these libraries and tools need to be installed just once, if you've Jupyter Notebook with C++, you can safely ignore this section.

- Install Miniconda: https://conda.io/miniconda.html
- open a terminal/shell and run the following commands
- create a virual environment to keep C++ specific installtions seperate from base installation

```bash
    conda create -n cpp python=3.7 # create virtual env named cpp with Python3.7 support
    conda activate cpp #activate the virual environemnt
    conda install notebook
    conda install -c conda-forge xeus-cling
    conda install -c conda-forge jupyter_contrib_nbextensions
    conda install -c conda-forge jupyter_nbextensions_configurator
    jupyter nbextensions_configurator enable --user
```

Run notebooks
- clone the repository locally once the tools are installed

- open a terminal and cd into this cloned repo and run jupyter notebook

```bash
    cd <cs2notebooks repo folder>
    jupyter notebook
```

- Enter ctrl+c to stop jupyter notebook from the terminal where its running from
- $ conda deactivate # to deactivate the virtual env and go back to base installation

1.8 output formatting

- iomanip - https://en.cppreference.com/w/cpp/header/iomanip
- hex, oct, fixed and scientific formats to display float values
- showpoints

[11]:
```
cout << fixed << setprecision(2) << 19.19999 << endl;
cout << hex << showbase << "hex 16 = " << 16 << " oct 8 = " << oct << 8 << endl␣
 ↪<< dec;
cout << setfill('*') << setw(10) << right << "hi " << left << setw(15) <<␣
 ↪"there!" << endl;
```

```
19.20
hex 16 = 0x10 oct 8 = 010
*******hi there!*********
```

1.9 functions

- sub-routine/sub-program/procedure
- burrowed from math/algebra concept: $y = f(x) = 2x^2 + 3x + 10$
- block of code identified by a single identifier
- two steps:
    1. define function
    2. call function
- defination syntax:

```
type functionName(type1 para1, type2& para2, ...) {
    /* block of code */
    return;
}
```

- call syntax:

```
functionName(arg1, arg2, ...);
```

- helps break problems into sub-problems
- helps code reuse and code abstraction (hiding implementation details)
- function can call many other functions

- two ways to pass data to a function: by value and by reference
- fruitful function can return answer/value from function
  - fruitful functions can be automatically tested
- void functions do not return a value; values printed are usually manually tested

[12]:
```
int add(int a, int b) {
    return a+b;
}
```

[13]:
```
int num1 = 100;
int num2 = -50;
cout << add(num1, num2) << endl;
```

```
50
```

1.10 unittest

https://en.cppreference.com/w/cpp/error/assert - automatic testing of functions - use assert function defined in assert.h or cassert header file

[14]:
```
#include <cassert>
//#include <assert.h>
assert(add(99, 1) == 100);
```

[15]:
```
assert(add(100, 200) == 400); // this should give assertion error but doesn't␣
↪work here...
// try it here: https://coliru.stacked-crooked.com/
```

1.11 string data

https://en.cppreference.com/w/cpp/string/basic_string - two ways to work with string: 1. C string - array of char type 2. C++ string - Abstract Data Type (ADT);

1.11.1 C string

- array of characters
- must be \0 (null terminated) to prevent from buffer-overrun
- many limitations while manipulating c string
- must learn to mitigate buffer overflow vulnerabiliy

```
char name[size];
```

[16]:
```
#include <cstring>
```

[17]:
```
// declare c-string
char name[20];
```

[18]:
```
strncpy(name, "John\0", 5);
cout << name << endl;
```

```
cout << strlen(name) << endl;
```

```
John
4
```

[19]:
```
// declare and initialize c-string
char word[] = "Hello";
```

[20]:
```
cout << word << endl;
cout << "len of word = " << strlen(word) << endl;
```

```
Hello
len of word = 5
```

1.11.2 C++ string

- must include string header file
- not fundamental type; but ADT (Abstract Data Type); user-defined type that's part of library

```
string varName;
```

[21]:
```
string phrase;
```

[22]:
```
phrase = "There may be a needle in the stack of stack of haystacks!";
```

[23]:
```
cout << phrase.length() << endl;
cout << phrase[0] << endl;
// cout << phrase.find("needle", 0);
// https://en.cppreference.com/w/cpp/string/basic_string/find
```

```
57
T
```

[23]:
```
@0x108b80ec0
```

[24]:
```
// loop throuch each char at a time
for (auto c: phrase)
    cout << c << " ";
```

```
T h e r e   m a y   b e   a   n e e d l e   i n   t h e   s t a c k   o f   s t
a c k   o f   h a y s t a c k s !
```

[25]:
```
// declare and initialize
string phrase1 = "Another phrase!"
```

1.12 escape sequences

https://en.cppreference.com/w/cpp/language/escape - used to represent certain special characters within string literals

character	description
\\	backslash
\’	single quote
\”	double quote
\n	new line - line feed
\r	carriage return
^	horizontal tab
\v	vertical tab
\b	backspace

[26]:
```
char q = '\'';
cout << q;
```

```
'
```

[27]:
```
string sent = "\"Oh no!\", exclaimed Jill. \"Jack broke my bike!\"";
cout << sent;
```

```
"Oh no!", exclaimed Jill. "Jack broke my bike!"
```

1.13 Operators

- operators and precedence rule: https://en.cppreference.com/w/cpp/language/operator_precedence
- arithmetic operators: https://en.cppreference.com/w/cpp/language/operator_arithmetic

1.14 unary operators

Operator	Symbol	Syntax	Operation
positive	+	+100	positive 100 (default)
negative	-	-23.45	negative 23.45

1.15 binary operators

- take two operands
- follows PEMDAS rule of precedence

Operator	Symbol	Syntax	Operation
add	+	x + y	add the value of y with the value of x
subtract	-	x - y	subtract y from x
multiply	*	x * y	multiply x by y
divide	/	x / y	divide x by y (int division if x and y are both ints)
modulo	%	x % y	remainder when

1.16 binary bitwise operators

- https://www.learncpp.com/cpp-tutorial/38-bitwise-operators/

Operator	Symbol	Syntax	Operation
bitwise left shift	<<	x << y	all bits in x shifted left y bits; multiplication by 2
bitwise right shift	>>	x >> y	all bits in x shifted right y bits; division by 2
bitwise NOT	~	~x	all bits in x flipped
bitwise AND	&	x & y	each bit in x AND each bit in y
bitwise OR	|	x | y	each bit in x OR each bit in y
bitwise XOR	^	x ^ y	each bit in x XOR each bit in y

[28]:
```
cout << " bitwise and &" << endl;
cout << (1 & 1) << endl;
cout << (1 & 0) << endl;
cout << (0 & 1) << endl;
cout << (0 & 0) << endl;
```

```
 bitwise and &
1
0
0
0
```

[29]:
```
cout << "bitwise or | " << endl;
cout << (1 | 1) << endl;
cout << (1 | 0) << endl;
cout << (0 | 1) << endl;
cout << (0 | 0) << endl;
```

```
bitwise or |
1
1
1
0
```

[30]:
```
cout << "bitwise not ~" << endl;
cout << ~(1|1) << endl;
cout << ~0 << endl;
```

```
bitwise not ~
-2
-1
```

[31]:
```
cout << "bitwise xor ^" << endl;
cout << (1 ^ 1) << endl;
cout << (1 ^ 0) << endl;
cout << (0 ^ 1) << endl;
cout << (0 ^ 0) << endl;
```

```
bitwise xor ^
0
1
1
0
```

[33]:
```
cout << (1 << 10); // pow(2, 10)
```

```
1024
```

[33]:
```
@0x108b80ec0
```

[34]:
```
cout << (1024 >> 10);
```

```
1
```

1.17 Ternary conditional operator (? :)

- syntax:

```
(condition) ? TrueValue : FalseValue;
```

[35]:
```
int number1, number2, larger;
```

[36]:
```
number1 = 10;
number2 = 20;
```

[37]:
```
larger = (number1 > number2) ? number1 : number2;
cout << "larger = " << larger << endl;
```

```
larger = 20
```

1.18 Other operators

- scope resolution operator: ::

  - std::string, std::cin
  - can create your own namespace : http://www.cplusplus.com/doc/tutorial/namespaces/
- increment/decrement (pre and post): ++, –
- compound assignments: +=, -=, *=, /-, %=, <<=, >>=, &=, ^=, |=

1.19 Math functions

- cmath library for advanced math operations: https://en.cppreference.com/w/cpp/header/cmath
  - ceil, floor, round, sqrt, pow, abs, log, sin, cos, tan

1.20 conditionals - control flow

- select a block of code to execute based on some condition
- add logic to the code as if your code is thinking and making a decision
- use boolean expression that evaluates to true or false
- use comparision operators (== , != , <=, >=) to compare values/expressions that will provide true or false or (yes or no) result
- use logical operators (&&, ||) to formulate compound logical expression
- three types:
  1. one-way selector
  2. two-way selector
  3. multi-way selector
- one selector type can be nested inside another!

1.20.1 one way selector

```
if (expression == true) {
  /* execute code... */
}
```

[38]:
```
// one way selector
bool execute = false;
if (execute)
    cout << "this block executed!" << endl;

cout << "done!"
```

```
done!
```

[38]:
```
@0x108b80ec0
```

1.20.2 two way selector

```
if (expression == true) {
    /* execute this block */
}
else {
    /* otherwise, execute this block */
}
```

[39]:
```
// two way selector
// test if a given number is even or odd
bool isEven(int n) {
    if (n%2 == 0)
        return true;
    else
        return false;
}
```

[40]:
```
int someNum;
```

[41]:
```
someNum = 11;
if (isEven(someNum))
    cout << someNum << " is even!";
else
    cout << someNum << " is odd!";
```

```
11 is odd!
```

1.20.3 multi-way selector

```
if (expression1 == true) {
    /* execute this block and continue after else block */
}
else if (expression2 == true) {
    /* execute this block and continue after else block */
}
else if (expression3 == true) {
    /* execute this block and continue after else block*/
}
...
else {
   /* if no condition is evaluated true, by default execute
   this block
   */
}
```

[42]:
```
int day;
```

[43]:
```
// multiway selector
day = 0;
if (day == 0) {
    cout << "Sunday" << endl;
    cout << "Yay!! it's a weekend!\n";
}
else if (day == 1)
    cout << "Monday";
else if (day == 2)
```

```
    cout << "Tuesday";
else if (day == 3)
    cout << "Wednesday";
else if (day == 4)
    cout << "Thursday";
else if (day == 5) {
    cout << "Friday";
    cout << "Almost weekend!";
}
else {
    cout << "Saturday" << endl;
    cout << "Yay!! it's a weekend!\n";
}
cout << "done..." << endl;
```

```
Sunday
Yay!! it's a weekend!
done…
```

1.20.4 swtich statment

https://en.cppreference.com/w/cpp/language/switch - works on integral data/expression to compare its value with many cases - similar to multi-way selector are more efficient and adds readability

```
switch (expression) {
    case value1:
        //...
        break;
    case value2:
    case value3:
        //...
        break;
    default:
        //...
}
```

[44]:
```
enum Colors
{
    COLOR_BLACK,
    COLOR_WHITE,
    COLOR_RED,
    COLOR_GREEN,
    COLOR_BLUE
};
```

[45]:
```
void printColor(Colors color) {
    switch (color) {
        case COLOR_BLACK:
```

```
            std::cout << "Black";
            break;
        case COLOR_WHITE:
            std::cout << "White";
            break;
        case COLOR_RED:
            std::cout << "Red";
            break;
        case COLOR_GREEN:
            std::cout << "Green";
            break;
        case COLOR_BLUE:
            std::cout << "Blue";
            break;
        default:
            std::cout << "Unknown";
            break;
    }
}
```

[46]:
```
Colors favColor;
```

[47]:
```
favColor = COLOR_BLACK;
printColor(favColor);
```

```
Black
```

1.21 loops - control flows

- repeatedly execute a block of code over-again with a different result
- be careful of infinite loop!
- `break` and `continue` keywords can be used inside loop
  - `break` : breaks the loop immidiately ignoring all the trailing codes and execution continues after loop body
  - `continue` : continues to the next iteration ignoring all the trailing codes inside loop
- four types of loop structures
  1. for loop
  2. range based for loop
  3. while loop
  4. do while

1.21.1 for loop

https://en.cppreference.com/w/cpp/language/for

```
for(init; condition; update) {
    // statements
}
```

- order of execution:
  1. init; only one
  2. condition
  3. statements
  4. update
  5. repeat from step 2

[48]:
```
for (int i=1; i<=20; i++) {
    if (i%2 == 0)
        cout << i << " ";
}
```

```
2 4 6 8 10 12 14 16 18 20
```

1.21.2 range-based for loop

https://en.cppreference.com/w/cpp/language/range-for

```
for(range_declaraion: range_expression) {
    // statements
}
```

[49]:
```
for (auto num: {1, 2, 3, 4, 100})
    cout << num << " ";
```

```
1 2 3 4 100
```

[56]:
```
string hello = "Hello World";
```

[57]:
```
for (char ch: hello)
    cout << ch << "-";
```

```
H-e-l-l-o- -W-o-r-l-d-
```

1.21.3 while loop

https://en.cppreference.com/w/cpp/language/while - Executes statement(s) repeatedly, until the value of condition becomes false. - The test takes place before each iteration.

```
while (condition) {
    // statements
}
```

[2]:
```
int k;
```

[3]:
```
k = 0;
while (k <= 20) {
    if (k%2 == 0)
        cout << k << " ";
    k += 1; // DO NOT FORGET TO UPDATE LOOP VARIABLE TO AVOID INFINITE LOOP!!!
```

```
}
```

```
0 2 4 6 8 10 12 14 16 18 20
```

1.21.4 do while loop

https://en.cppreference.com/w/cpp/language/do - executes statement(s) repeatedly, until the value of condition becomes false. - the test takes place after each iteration.

```
do {
    // statements
} while (condition);
```

[4]:
```
int MAX_TIMES;
int times;
```

[5]:
```
times = 0;
MAX_TIMES = 20;
do {
    cout << times << " ";
    times ++; // DO NOT FORGET TO UPDATE LOOP VARIABLE TO AVOID INFINITE LOOP!!!
    break;
} while(times <= MAX_TIMES);
```

```
0
```

1.22 arrays

- container that stores 1 or more similar data called elements
- use array when you need to store large number of data values as declaring individual variable for each variable is not desireable or not even possible!
- size of the array has to be known and is fixed
- use 0-based index to access each element stored in array
- array is passed by reference ONLY to a function
- array can't be returned from a function
- aggregate operations such as IO, assignment, comparision are not allowed
- syntax:

```
type arrayName[const_size];
```

[6]:
```
int tests[10];
float prices[] = {1, 2, 3, 100.99};
string names[2] = {"John", "James"};
```

[7]:
```
tests
```

[7]:
```
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
```

[8]:
```
prices
```

[8]:
```
{ 1.00000f, 2.00000f, 3.00000f, 100.990f }
```

[9]:
```
names
```

[9]:
```
{ "John", "James" }
```

[10]:
```
tests[0] = 100;
tests[9] = 75;
tests
```

[10]:
```
{ 100, 0, 0, 0, 0, 0, 0, 0, 0, 75 }
```

[11]:
```
prices = {2, 3, 4, 5}
```

```
input_line_26:2:9: error: array type 'float [4]' is not
assignable
 prices = {2, 3, 4, 5}
 ~~~~~~ ^
```

```
        Interpreter Error:
```

[12]:
```
cout << prices[10] << endl;
```

```
input_line_27:2:10: warning: array index 10 is past the
end of the array (which contains 4 elements) [-Warray-bounds]
 cout << prices[10] << endl;
         ^      ~~

input_line_14:3:1: note: array 'prices' declared
here
float prices[] = {1, 2, 3, 100.99};
^
```

```
2.2631e-34
```

1.23 2-D arrays

- row-major 2-d arrays
- syntax:

```
type name[rowSize][colSize];
```

[13]:
```
int matrix[4][4] = {{1, 2, 3, 4}, {10, 20, 30, 40}, {100, 200, 300, 400}, {1,␣
↪1, 1, 1}};
```

[14]:
```
matrix
```

[14]:
```
{ { 1, 2, 3, 4 }, { 10, 20, 30, 40 }, { 100, 200, 300, 400 }, { 1, 1, 1, 1 } }
```

[15]:
```
matrix[0][0] = matrix[2][0]*matrix[3][0]
```

[15]:
```
100
```

[16]:
```
matrix
```

[16]:
```
{ { 100, 2, 3, 4 }, { 10, 20, 30, 40 }, { 100, 200, 300, 400 }, { 1, 1, 1, 1 } }
```

[17]:
```
for(int i=0; i<4; i++) {
    cout << "[ ";
    for(int j=0; j< 4; j++) {
        cout << matrix[i][j] << ", ";
    }
    cout << "]" << endl;
}
```

```
[ 100, 2, 3, 4, ]
[ 10, 20, 30, 40, ]
[ 100, 200, 300, 400, ]
[ 1, 1, 1, 1, ]
```

[ ]:
```

```