DataVariablesAndOperations

July 16, 2021

1 Data, Variables and Operations

1.1 Topics

- data and values
- C++ fundamental data types
- digital units and number systems
- variables and data assignment
- keywords and operators
- order of operations
- operators for numbers and strings
- constants
- type casting

1.2 Data and values

- data and values are the fundamentals to any computer language and program
- a value is one of the fundamental things – like a letter or a number – that a program manipulates
- almost all computer programs use and manipulate some data values

1.3 Literal values and representations

- at a high level, we deal with two types of data values: Numbers and Texts
- numbers can be further divided into two types:
  - Whole number literal values: 109, -234, etc.
  - Floating point literal values: 123.456, -0.3555, etc.
- text is a collection of 1 or more characters (symbols, digits or alpahabets)
  - single character is represent using single quote (')
    - char literal values: 'A', 'a', '%', '1', etc.
  - 2 or more characters are called string
    - represented using double quotes (")
    - string literal values: "CO", "John Doe", "1100", etc.
- programming languages need to represent and use these data correctly

1.4 C++ fundamental types

- there are many fundamental types based on the size of the data program needs to store

- most fundamental types are numeric types
- see here for all the supported types: https://en.cppreference.com/w/cpp/language/types
- the most common types we use are:

Type	Description	Storage size	Value range
void	an empty set of values; no type	system dependent: 4 or 8 bytes	NA
bool	true or false	1 byte or 8 bits	true or false 1 or 0
char	one ASCII character	1 byte or 8 bits	-2^7 to $2^7 - 1$
unsigned char	one ASCII character	1 byte or 8 bits	0 to $2^8 - 1$
int	+/-ve integers	4 bytes	-2^{31} to $2^{31} - 1$
unsigned int	only positive integers	4 bytes or 32 bits	0 to $2^{32} - 1$
long	+ve and -ve big integers	8 bytes or 64 bits	-2^{63} to $2^{63} - 1$
unsigned long	positive big integers	8 bytes or 64 bits	0 to $2^{64} - 1$
float	single precision floating points	32 bits	7 decimal points
double	double precision floating points	64 bits	15 decimal points

- in C++, there's no fundadamental type available to work with string data
- two common ways to store string data:
  - use C-string or array of characters
  - use basic_string defined in `<string>` library
    - more on basic_string: https://en.cppreference.com/w/cpp/string/basic_string
    - must include `<string>` library and **std** namespace
- we'll dive into string more in depth in **Strings** chapter

1.4.1 sizeof operator

- one may want to know the size of memory allocated for the fundamental types
  - some of these types are system dependent (e.g., long is 32 bit in x86 and 64 bit in x64)
- **sizeof(type)** operator gives size of fundamental types in bytes
- let's check the size of some fundamental types on my 64-bit MacBook Pro laptop

[1]:
```
sizeof(bool)
```

[1]:
```
1
```

[2]:
```
sizeof(char)
```

[2]:
```
1
```

[3]:
```
sizeof(int)
```

[3]:
```
4
```

[4]:
```
sizeof(long)
```

[4]:
```
8
```

[5]:
```
sizeof(float)
```

[5]:
```
4
```

[6]:
```
sizeof(double)
```

[6]:
```
8
```

1.5 Units of digital data

- digital computers use binary number system consisting of two digits (0 and 1)
- every data and code is represented using binary values
  - hence the name binary or byte code for executable programs
  - letter A is encoded as 1000001 (7 binary digits)
- humans use decimal number system with 10 digits (0 to 9)
  - we have ways to represent texts using alphabets for English language e.g.: Hello Bond 707!
  - texts must be encoded into numbers, if we lived in the world that only understood numbers
- the following table shows the various units of digital data

Unit	Equivalent
1 bit (b)	0 or 1
1 byte (B)	8 bits (b)
1 kilobyte (KB)	1,024 B
1 megabyte (MB)	1,024 KB
1 gigabtye (GB)	1,024 MB
1 terabyte (TB)	1,024 GB
1 petabyte (PB)	1,024 TB
...	...

1.6 Number systems

- there are several number systems based on the base digits
  - base is number of unique digits number system uses to represent numbers
- binary (base 2), octal (base 8), decimal (base 10), hexadecimal (base 16), etc.

1.6.1 Decimal number system

- also called Hindu-Arabic number system
- most commonly used number system that uses base 10
  - has 10 digits or numerals to represent numbers: 0..9
  - e.g., 1, 79, 1024, 12345, etc.
- numerals representing numbers have different place values depending on position:
  - ones (10^0), tens(10^1), hundreds(10^2), thousands(10^3), ten thousands(10^4), etc.
  - e.g., 543.21 = $(5 \times 10^2) + (4 \times 10^1) + (3 \times 10^0) + (2 \times 10^{-1}) + (1 \times 10^{-2})$

1.7 Number system conversion

- since computers understand only binary, everything (data, code) must be converted into binary
- all characaters (alphabets and symbols) are given decimal codes for electronic communication
  - these codes are called ASCII (American Standard Code for Information Interchange)
  - $A \to 65; Z \to 90; a \to 97; z \to 122, * \to 42$, etc.
  - see ASCII chart: https://en.cppreference.com/w/c/language/ascii

1.7.1 Converting decmial to binary number

- algorithm steps:
  1. repeteadly divide the decimal number by base 2 until the quotient becomes 0
     - note remainder for each division
  2. collect all the remainders in reverse order
     - the first remainder is the last (least significant) digit in binary
- example 1: what is decimal $(10)_{10}$ in binary $(?)_2$?
  - step 1:
    $\frac{10}{2}$ = quotient: 5, remainder: 0 $\frac{5}{2}$ = quotient: 2, remainder: 1 $\frac{2}{2}$ = quotient: 1, remainder: 0 $\frac{1}{2}$ = quotient: 0, remainder: 1
  - step 2:
    - collect remainders from bottom up: 1010
  - so, $(10)_{10} = (1010)_2$
- example 2: what is decimal $(13)_{10}$ in $(?)_2$?
  - step 1:
    $\frac{13}{2}$ = quotient: 6, remainder: 1 $\frac{6}{2}$ = quotient 3, remainder: 0 $\frac{3}{2}$ = quotient: 1, remainder: 1 $\frac{1}{2}$ = quotient: 0, remainder: 1
  - step 2:
    - collect remainders from bottom up: 1101
  - so, $(13)_{10} = (1101)_2$

1.7.2 Converting binary to decimal number

- once the computer does the computation in binary, it needs to convert the results back to decimal number system for humans to understand
- algorithm steps:
  1. multiply each binary digit by its place value in binary
  2. sum all the products
- example 1: what is binary $(1010)_2$ in decimal $(?)_{10}$?
  - step 1:
    - $0 \times 2^0 = 0$
    - $1 \times 2^1 = 2$
    - $0 \times 2^2 = 0$
    - $1 \times 2^3 = 8$
  - step 2:
    - $0 + 2 + 0 + 8 = 10$
  - so, $(1010)_2 = (10)_{10}$

- example 2: what is binary $(1101)_2$ in decimal $(?)_{10}$?
  - step 1:
    * $1 \times 2^0 = 1$
    * $0 \times 2^1 = 0$
    * $1 \times 2^2 = 4$
    * $1 \times 2^3 = 8$
  - step 2:
    * $1 + 0 + 4 + 8 = 13$
  - so, $(1101)_2$ = $(13)_{10}$
- we got the same decimal vales we started from in previous examples
- food for thought: think how you'd go about writing a program to convert any positive decimal number into binary and vice versa!

1.8 Negative (signed) integers - Two's complement

- most common method of storing negative numbers on computers is a mathematical operation called Two's complement
- Two's complement of an N-bit number is defined as its complement with respect to 2^N
  - the sum of a number and its two's complement is 2^N
- e.g.: for the 3-bit binary number 010_2, the two's complement is 110_2
  - because $010_2 + 110_2 = 1000_2 = 2^3_{10}$
- Two's complement of N-bit number can be found by flipping each bit and adding one to it
- e.g. find two's complement of 010
  - Algorithm steps:
    1. flipped each bit; 0 is flipped to 1 and 1 is flipped to 0

    $010 \rightarrow 101$

    2. add 1 to the flipped binary

```
 101
  +1
-----
 110
```

1.8.1 Example: What is -3 decimal in 8-bit binary representation?

- convert 3_10 to an 8-bit binary
  - $3_{10} \rightarrow 00000011_2$

1. find Two's complement of 8-bit binary
   - $00000011_2 \rightarrow 11111100_2 + 1 = 11111101_2$
2. Sanity check:
   - $00000011_2 + 11111101_2 = 100000000_2 = 2^8_{10}$

- So, $-3_{10} = 11111101_2$ in an 8-bit representation

1.9 Exercise

1. Convert decimal integer 7 into binary with 16 bits.

2. Convert -7 decimal integer into binary with 16 bits.

1.10 Variables

- programs must load data values into memory to manipulate them
- data may be large and used many times during the program
  - typing the data values literally all the time is not efficient and fun
  - most importantly error prone due to typos
  - you may not even know that values may be if they're read from standard input, files, etc.
- variables are named memory location where data can be stashed for easy access and manipulation
- one can declared and use as many variables as necessary
- C++ is statically and strongly typed programming language
  - variables are tied to their specific data types that must be explictly declared when declaring variables

1.10.1 Variable declaration

- statements that create variables/identifiers to store some data values
- as the name says, value of variables can vary/change over time
- syntax:

```
type varName;
type varNam1, varName2, ...; //declare several variables all of the same type
```

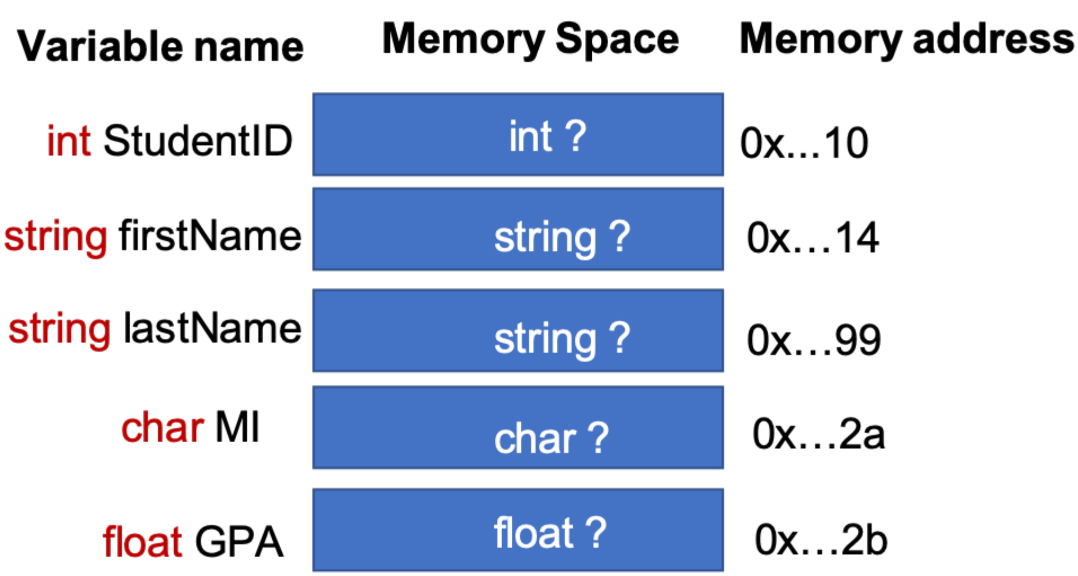

Fig. C++ Variables and Memory

1.10.2 Rules for creating variables

- variable names are case sensitive

- must declare variables before they can be used
- can't define variable with the same name more than once
- can't use keywords as variable names
- data stored must match the type of variable
- variable names can't contain symbols (white spaces, #, &, etc.) except for _ and $ (underscore and dollar)
- variable names can contain digits but can't start with a digit
- variable names can start with only alphabets (lower or upper) and _ symbol

1.10.3 Best practices

- use descriptive and meaningful but concise name
  - one should know quickly what data you're storing
- use lowercase; camelCase or (_ underscore) to combine multiple words

1.10.4 C++ keywords

- keywords are reserved names and words that have specific purpose in C++
  - they can only be used what they're intended for
- e.g., char, int, unsigned, signed, float, double, bool, if, for, while, return, struct, class, operator, try, etc.
- all C++ keywords are listed here: https://en.cppreference.com/w/cpp/keyword

[7]:
```
// examples of variable declaration
bool done;
char middleInitial;
char middleinitial; // hard to read all lowercase name
int temperature;
unsigned int age;
long richest_persons_networth;
float interestRate;
float length;
float width;
double space_shuttle_velocity;
```

[8]:
```
// TODO:
// Declare 10 variables of atleast 5 different types
```

1.10.5 String variables

- declare variables that store string data
  - 1 or more string of characters
- an easy way to use string is by using C++ advanced type defined in `<string>` header file
- must include `<string>` header file or library to use string type
- must also use **std** namespace
- strings are represented using a pair of double quotes ("string")
- more on string type is covered in **Strings** chapter
- the following are some examples of string variables

[9]:
```
// string variables
#include <string>

using namespace std;

string fullName;
string firstName;
string address1;
string country;
string state_name;
std::string state_code; // :: name resolution operator
```

[10]:
```
// TODO:
// Declare 5 string variables
```

1.11 Assignment operator (=)

- once variables are declared, data can be stored using assignment operator, \$ = \$
- **assignment statements** have the following syntax:

```
varName = value;
```

- since C++ is a strongly typed language, the type of value must match the type of variable
  - strongly typed languages enforces type safety and matching during the compile time

[11]:
```
// assignment examples
done = false;
middleInitial = 'J'; // character is represent using single quote
middleinitial = 'Q';
temperature = 73;
age = 45;
richest_persons_networth = 120000000000; // 120 billion
interestRate = 4.5;
length = 10.5;
width = 99.99f; // number can end with f to represent as float
space_shuttle_velocity = 950.1234567891234567; // 16 decimal points
```

[11]:
```
950.123
```

[12]:
```
// string assignment examples
fullName = "John Doe";
firstName = "John";
address1 = "1100 North Avenue"; // number as string
country = "USA";
state_name = "Colorado";
state_code = "CO";
```

[13]:
```
// TODO: assign different values to variables defined above
```

1.11.1 Variable declartion and initialization

- variables can be declared with initial value at the time of construction
- if you know what value a variable should start with; this saves you typing
- often times its the best practice to initialize variable with default value
- several ways to initialize variables: https://en.cppreference.com/w/cpp/language/initialization
- two common ways:
  1. Copy initialization (using = operator)
  2. Value initialization (using { } curley braces)
     - also called uniform initialization
     - useful in initializing advanced types such as arrays, objects, etc.

[14]:
```
// Copy initialization
float price = 2.99f;
char MI = 'B'; //middle initial
string school_name = "Grand Junction High";
```

[15]:
```
// Value/uniform initialization
char some_letter{'U'};
int some_length{100};
float some_float{200.99};
string some_string = {"Hello World!"}; // can also combine the two!
```

1.11.2 Variable's value can be changed

- variable's value can vary through out the program
  - hence the name variable
- however, type of the value must be same as the type of the variable declared
- C++ is a strongly and statically typed programming language!

[16]:
```
price = 3.99;
price = 1.99;
MI = 'Q';
school_name = "Fruita Monument High";
some_string = "Goodbye, World!";
```

[17]:
```
price = "4.99"; // is this valid?
```

```
input_line_34:2:10: error: assigning to 'float' from

incompatible type 'const char [5]'
 price = "4.99";
         ^~~~~~
```

```
Interpreter Error:
```

1.11.3 auto type

- if variable is declared and initialized in one statement, you can use **auto** keyword to let compiler determine type of variable based on the value it's initialized with

[18]:
```
auto var1 = 10; // integer
auto var2 = 19.99f; // float
auto var3 = 99.245; // double
auto var4 = '@'; // char
```

[19]:
```
// char * (pointer) type and not string type
auto full_name = "John Doe";
```

[20]:
```
// can automatically declare string type
#include <string>
using namespace std;

auto full_name1 = string("Jake Smith"); // string type!
```

[21]:
```
// use typeid function to find the name of the types
// typeid is defined in typeinfo library
#include <typeinfo>
```

[22]:
```
typeid(full_name1).name()
```

[22]:
```
"NSt3__112basic_stringIcNS_11char_traitsIcEENS_9allocatorIcEEEE"
```

[23]:
```
// should print "i" -> short for integer
// Note: may also print invalid memory address in Jupyter notebook!
typeid(var1).name()
```

[23]:
```
"i"
```

1.11.4 Visualize variables and memory with pythontutor.com

1.12 Operators

- special symbols used to represent simple computations
  - like addition, multiplication, modulo, etc.
- C++ has operators for numbers, characters, and strings
- operators and precedence rule: https://en.cppreference.com/w/cpp/language/operator_precedence
- arithmetic operators: https://en.cppreference.com/w/cpp/language/operator_arithmetic

1.12.1 Unary operators

- takes one operand
- operands are values that operators work on
- there are two unary operators for numeric operands

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

1.12.2 Binary operators

- binary operators take two operands (left operator right)
- the following table shows the binary operators for numeric operands

Operator	Symbol	Name	Syntax	Operation
add	+	plus	x + y	add the value of y with the value of x
subtract	-	hyphen	x - y	subtract y from x
multiply	*	asterick	x * y	product of x and y
divide	/	slash	x / y	divide x by y (int division if x and y are both ints)
modulo	%	percent	x % y	remainder when x is divided by y

1.12.3 Adding numbers

- + symbol is used to add literal values or variables

[24]:
```
// adding literal integer values
+1 + (-1)
```

[24]:
```
0
```

[25]:
```
// adding literal floating points
99.9 + 0.1
```

[25]:
```
100
```

[26]:
```
// adding int variables
int num1, num2, sum;
```

[27]:
```
num1 = 10;
num2 = 5;
sum = num1 + num2;
```

[28]:
```
// let's see the value of sum
sum
```

[28]:
```
15
```

[29]:
```
// adding float variables
float n1 = 3.5;
float n2 = 2.5;
float total = n1+n2;
```

[30]:
```
// see total values
total
```

[30]:
```
6f
```

1.12.4 Subtracting numbers

- \- symbol is used to subtract literal numbers or variables

[31]:
```
// subtracting literal integers
10-1
```

[31]:
```
9
```

[32]:
```
// subtracting literal floating points
99.99 - 10.99
```

[32]:
```
89
```

[33]:
```
// subtracting variables
num1-num2
```

[33]:
```
5
```

1.12.5 Multiplying numbers

- * asterick symbol is used to multiply literal numbers and variables

[34]:
```
// multiplying literal integers
2*3
```

[34]:
```
6
```

[35]:
```
// multiplying literatl floats
2.5 * 2.0
```

[35]:
```
5
```

[36]:
```
// multiplying numeric variables
n1*n2
```

[36]:
```
8.75f
```

1.12.6 Dividing numbers

- / symbol is used to divide literal numbers or variables

[37]:
```
// dividing literal integers
10/2
```

[37]:
```
5
```

[38]:
```
9/2 // integer division; remainder is discarded
```

[38]:
```
4
```

[39]:
```
// dividing literal floats
// if one of the operands is floating point number, C++ performs float division
9.0/2
```

[39]:
```
4.5
```

[40]:
```
// dividing numeric variables
n1/n2
```

[40]:
```
1.4f
```

1.12.7 Capturing remainder from a division

- use modulo or remainder (%) operator to find the remainder of literal values or variables
- only works on positive integers

[41]:
```
// modulo or remainder operator
5%2 // testing for odd number
```

[41]:
```
1
```

[42]:
```
4%2 // testing for even number
```

[42]:
```
0
```

[43]:
```
// can't divide 10 by 11
10%11
```

[43]:
```
10
```

[44]:
```
// expressions with variables and literals
// declare some variables
int hour, minute;
```

[45]:
```
// assign some values
hour = 11;
minute = 59;
```