# Summary of Section 2.1

## Declare a variable

A variable is declared exactly once. A variable must be declared before it can be used, where used means to print out its value, initialize (set value for the first time), change value, or being part of an expression.

There are two ways to declare a variable.

**type** variable\_name;

**type** variable\_name = value\_fit\_for\_that\_type;

Note that value needs to match type. For example, you cannot put an int value into a string variable or vice versa.

The following code illustrates on how to declare and initialize int and string variables.

//Declare int variable age and initialize it to be 18.

int age = 18;

//Declare string variable name.

string name;

//Initialize name variable to be “John Doe”.

name = "John Doe";

## Use assignment operator = to initialize or change values

Assignment operator = sets up value for a variable.

Assignment operator = is executed from right to left. First calculate the value in right-hand side expression, then use that value to set the left-hand side variable.

There are two ways to initialize a variable.

One way is to put a literal of its type to it like

int age = 18;

Another way is to enter value to variable using cin. We call this approach interactive input.

//cout is an object for character output.

//You may think cout as screen.

//Operator << pushes value to cout.

//It is user-friendly to prompt user what to input.

cout << "Enter your age: ";

//Declare int varible age without initialization.

int age;

//cin is an object for character input.

//You may think cin as keyboard.

//Operator >> pulls contents from cin to variable.

//Initialize age by taking its value interactively from cin.

cin >> age;

cout << "Enter your name: ";

string name;

//cin >> name;

//only extract the first word

//(letter sequence before the first space or return key,

//whichever is first met) to name.

//Suppose you enter "John Doe" (without quotients) and

//return key, then name is set to be "John".

cin >> name;

Here are some more examples.

int age = 18;

age = age + 3; //increase age by 3.

//After this statement, age is 21.

age++; //increase age by 1, age is 22 now.

age += 2; //same as age = age + 2; increase age by 2.

## Naming of variable

# A variable in C++ starts with a letter or underscore (\_), followed by letter, underscore (\_), or digits. It cannot be a reversed word in C++.

# A variable name needs to be meaningful. If a name involves two or more words, this book follows C++ STL (Standard Template Library) and uses \_ to separate meaningful words.

# For example, to represent the area of a circle, we would like to use *circle\_area* instead of *circlearea*, the latter is hard to read. If there is no confusion, you can also name it *area*.

Some people prefer to use camel case naming. That is, the first letter of each meaningful word (except the first one) is capitalized. For example, use *circleArea* to name circle area. That is OK,

If a variable is a constant, then by convention use all capital letters. For example,

const int MINUTES\_PER\_HOUR = 60;

## Variable types

To represent whole number, we normally use **int** type. Other types can be **signed, unsigned, short, long**, and so on. The difference is the size and the range of integers can be represented.

To represent number with decimal parts, we normal use **double**, which means double-precision floating point numbers.

To represent a character, we use **char** type. For example, declare letter grade is ‘A’, we use

char letter\_grade = 'A';

Note that a character liter is enclosed in a pair of matched single quotes. A string literal is enclosed in a pair of double quotes.

## Range of a type

Due to memory size limit, one cannot represent infinite numbers. For example, an int in C++ can represent ints in [-231, 231-1] or -2,147,483,648 to 2,147,483,647.

Pay attention to range of data.

int num = 1000000000; //1000000000 is one billion.

//cannot write 1000000000 as 1,000,000,000.

    //comma representation is not allowed in C++.

num \*= 3; //same as num = num \* 3;

          //That is, triple the value of num.

cout << "num = " << num << endl;

//The output is num = -1294967296, which is not correct.

//reason: 3 billion is out of range of int.

Here is a fix. Change the type of num from int to long. Type long has 64-bit and can represent larger range of integers than int.

long num = 1000000000;

num \*= 3;

cout << "num = " << num << endl; //output num = 3000000000

## Constant and its application

If a variable always keeps the same value, we put **const** before its declaration. It is similar to a dumb bell, which always keeps its factory setting.

When naming a constant variable, by convention, we use all capital letters.

Use const makes code clear and easier to maintain. For example, suppose we declare PI as const variable and initialize its value as 3.14. Later on, if we want to be more precise, we change its value to 3.14 to 3.1415, this only needs one modification in const variable declaration.

double PI = 3.14;

double r = 2;

double circle\_perimeter = 2 \* PI \* r;

double circle\_area = PI \* r \* r;

double sphere\_volume = 4.0 / 3 \* PI \* r \* r \* r;

//cannot use 4.0 / 3 by 4 / 3

//since 4 / 3 is integer division and returns 1.

# As an aside, if you import cmath library by

#include<cmath>

Then you can use M\_PI to save the headache of defining PI yourself.

Type double representation may not be precise

Due to memory size limit and some double numbers can have infinite number of digits, calculation involving double may not be 100% precise.