CH 3: Expressions and interactivity

3.1: cin Object

cin reads input from the console (or the keyboard).

The operator >> is called _____ operator.

The cin object causes a program to _____ at the keyboard and the Enter key is pressed. No other lines in the program will be executed until cin gets its input.

cin automatically converts the data read from the keyboard to the data type of the variable used to store it.

Write a C++ statement to read an integer for the age from the keyboard. (You may assume the following definition already have made: int age;).

Code:

cout << "Please enter the age you wish you could permanently be? " << endl;

.....

Output:

Please enter the age you wish you could permanently be?
25 (Enter]

Write statement(s) to read three whole numbers from the keyboard and store them in separate variables; you may assume the variable definition:

int test1Score,

test2Score,

test3Score;

method 1 (on one line; one statement):

.....

.....

method 2 (three lines; one statement):

.....

.....

.....

method 3 (on three lines; three statements):

.....

.....

.....

Write ONE C++ statement to read three real numbers from the keyboard and store them in separate variables. You may assume the following variable definitions:

double temp1,
temp2,
temp3;

cout << "Enter the last three temperature readings." << endl;

.....

.....

.....

Output:

Enter the last three temperature readings.

101.6 100.2 98.7 [Enter]

What would be the output from the following statements which are placed below the above code in your program. ts:

cout << temp1 << " " << temp2 << " " << temp3 << endl;

Output:

.....

.....

An expression is a programming statement that has a value. (note: there is only one and only one variable on the left-hand side of the equal sign.)

e.g.,

```
sum = 21 + 3;                    //24 is stored in sum
number = 4;
number = x;
```


```
double num1 = 5,
       num2 = 2,
       prod;

prod = num1 * num2;
cout << "product  = " << prod;
```

Output:

.....

Mathematical expression is with the cout object (**Warning!!!!! This is bad programming.)**

```
double num1 = 5,
       num2 = 2;
cout << "product  = " << = num1 * num2;
```


Output:

.....

From Chapter 2: Modulus

Modulus works only with Integer operands.

It returns the remainder of an integer division.

```
      4
3 | 1  3
    1  2
    ---
    **1**
```

13 % 3 = 1

e.g.,

10 % 3 = 1

11 % 3 = 2

12 % 3 = 0

13 % 3 = 1

10 % 4 = 2

11 % 4 = 3

12 % 4 = 0

How do you find a number even or odd?

```
int number;
cout << "Enter a number" << endl;
cin << number;
remainder = number % 2;
```

conclusion:

if the remainder is ..... then the number is: .....

How do you find a number is divisible by 3 or not?

```
int number;
cout << "Enter a number" << endl;
cin << number;
remainder = number % _____;
```

conclusion:

if the remainder is ..... then the number is: .....

HW 4: (Submit through WorldClassroom)

Complete the following program that calculates the change in terms of different types of coins using minimum total number of coins.

```
int main()
{
    int balance,                                        // hold the balance in cents
        quarters,
        dimes,
        nickels,
        pennies,
        remainder;
    const int QUATER = 25,
              DIME = 10,
              NICKEL = 5;

    cout << "Enter the balance in cents: ";
    cin >> balance;

    quarters = balance / QUATER;                //Find #of quarters

    remainder = ........................        //Find the remaining balance


    dimes = ............................        //Find the #of dimes

    remainder = ........................        //Find the remaining balance


     ...................................        //Find the #of Nickels

    ...................................         //Find the remaining balance


    ...................................         //remaining balance would be all pennies


    cout << "Balance" << balance << endl

       ..........................................................................

       ..........................................................................

       ..........................................................................

      ...........................................................................

    return 0;

}
```

Operator Precedence

Precedence of arithmetic operators

Precedence of arithmetic operators	Associativity of arithmetic Operator
- (unary negation)	Right to left
* / %	Left to right
+ -	Left to right

e.g.,

-9 + 7 % 3

= -9 +1

= -8

53 + 7 - 25

= 60 - 25

= 35

Grouping with parenthesis

Parts of a mathematical expression may be grouped with parenthesis to force some operations to be performed before others.

e.g.,

9 + 6 * 3

= 9 + 18

= 27

Vs

(9 + 6) * 3

= 15 * 3

= 45

Expression	Show your work here	Value of x
x = (5 + 2) * 4;		
x = 10 / (5 - 3);		
x = 8 + 12 * (6 – 2);		
x = (4 + 17) % 2 - 1 ;		
x = (6 – 3) * (2 + 7) / 3;		

Converting Algebraic expressions to programming Statements; store the value in a variable called ***ans***.

Expression in Algebra	Expression in C++	Notes
6b	ans = 6 * b;	
5rs		
$\frac{4}{3}\pi r^2$		
$\frac{b^2 - 4ac}{2a}$		

Power function

e.g., πr^2 = PI * pow (r, 2.0)

Notes:

Must include a preprocessor directive named cmath; i.e., ……..

The arguments should be defined as ……

The value returned (in simple words, the answer) should be stored in ……..

preprocessor directives learnt so far:

#include <iostream>	To use cin and cout objects
#include <string>	To use string objects.
#include <cmath>	To use mathematical functions, such as pow

Write the code to get the result of r^2.

…….. r,
answer;

answer = pow (r, 2.0);

Type Conversion

Code:

```
int numerator = 5,
    denominator = 2;

cout << "Fraction  = " << numerator / denominator;
```

Output:

.....

Conclusion:

.....

Code:

```
double numerator =5;
int denominator =2;

cout << "Fraction  = " << numerator / denominator;
```

Output:

.....

Conclusion:

.....

Data Type Ranking

Typical Data Type Sizes

data type	Typical size
short int	2
unsigned short int	2
int	4
unsigned int	4
long int	4
unsigned long int	4
long long int	8
unsigned long long int	8
float	4
double	8
long double	8

Data Type Ranking

data type	Typical size
long double	8
double	8
float	4
unsigned long long int	8
long long int	8
unsigned long int	4
long int	4
unsigned int	4
int	4
unsigned short int	2
short int	2
char	1

Data Promotion and Data Demotion

Promotion: When a value is converted to a ..... data type, it is said to be promoted.

Demotion: When a value is converted to a ..... data type, it is said to be demoted.

e.g., promotion

Code:

```
double numerator = 5,
int denominator = 2;
cout << “Fraction  = ” << numerator / denominator;
```

Output:

.....

Conclusion

.....

e.g., demotion

Code:

```
double temperature = 99.99;
int temp;
temp = temperature;
cout << “temperature =” << temperature;
cout << “temp =” << temp;
```

Output:

.....

.....

Conclusion:

.....

Rules:

1. char, short int, and unsigned short int are ..... to int.

   In some systems, unsigned short might take 4 bytes similar to int. In that case, unsigned short is promoted to unsigned int.

2. When an operator works with two different data types, the ..... ranking value is promoted to the type of the higher ranking value.
3. When the final value of an expression is assigned to a variable, it will be converted to the data type of the ..... to which the value is assigned to.

```
short shortNum = 3;
char grade = 'A';

cout << "1) " << sizeof(shortNum) << endl;      // short
cout << "2) " << sizeof(2) << endl;             // int
cout << "3) " << sizeof(4.0) << endl;           //double
cout << "4) " << sizeof(grade) << endl;         //char

cout << "5) " << grade << endl << endl;         // the value in grade

cout << "6) " << sizeof(2 * shortNum) << endl; // int * short
cout << "7) " << sizeof(2 * 4.0) << endl;       // int * double

cout << "8) " << 2 * grade << endl;             // int * char
cout << "9) " << 2.5 * grade << endl;           // double * char
```

output

.....

.....

.....

.....

.....

.....

.....

.....

.....

Overflow and underflow

When a variable is assigned a value that is too large (Overflow) or too small (underflow) in range for that variable's data type, the variable overflows or underflows.

```
unsigned short int num1 = USHRT_MAX,
                   num2 = 10,
                   num3;

num3 = num1 * num2;
 cout << num1 << " * " << num2 << " = " << num3;
```

Output:

65535 * 10 = 65526 .....

Conclusion

.....

```
 short int num1 = SHRT_MIN,
           num2 = -10,
           num3;

num3 = num1 + num2;
cout << num1 << " * " << num2 << " = " << num3 << endl;
```

Output:

-32768 * -10 = 32758 .....

Conclusion:

.....

Type Casting

Type cast expression lets you manually promote or demote a value.

RECALL:

```
double pizzaPerPerson;
int numParticipant = 20,
    numPizzas = 5;
pizzaPerPerson = numPizzas  / numParticipant;

cout << “pizza per Person =  ” << pizzaPerPerson << endl;
```

output:

.....

..... at least one of the int values to double or float using C++ style:
static_cast<float> (numPizzas)

```
pizzaPerPerson =  ..... / numparticipant;
```

RECOMMENDED!!!!

RECALL: INTEGER DIVISION

```
int numParticipant = 20,
    numPizzas = 5;

pizzaPerPerson = numPizzas  / numParticipant;

cout << “pizza per Person =  ” << pizzaPerPerson << endl;
```

output:

.....

How to get rid of integer division? Casting!!!

CASTING numerator

```
int numParticipant = 20,
    numPizzas = 5;

pizzaPerPerson = static_cast <float> (numPizzas)  / numParticipant;

cout << "pizza per Person =  " << pizzaPerPerson << endl;
```

output:

.....

CASTING denominator

```
int numParticipant = 20,
    numPizzas = 5;

pizzaPerPerson = numPizzas  / static_cast <float> (numParticipant);

cout << "pizza per Person =  " << pizzaPerPerson << endl;
```

output:

.....

A BIG COMMON MISTAKE

```
int numParticipant = 20,
    numPizzas = 5;

pizzaPerPerson = static_cast <float> (numPizzas  / numParticipant);

cout << "pizza per Person =  " << pizzaPerPerson << endl;
```

output:

.....

More examples: of casting.

CASTING a number to char

```
int number = 65,
    number2 = number + 1;

cout << static_cast <char> (number) << endl;
cout << static_cast <char> (number2) << endl;
```

output:

.....

.....

CASTING a char to an int

```
int character = 'A';

cout << static_cast <int> (character) << endl;
```

output:

.....

3.6 Multiple Assignment and Combined Assignment

Multiple Assignment: Assign the same value to several variables with one statement.

a = b = c = d = 12;

Combined Assignment

Assume x = 6

Statement	What it does	Value of x after the statement
x = x + 4;	add 4 to x and store back in x	
x = x - 3;	subtract 3 from x and store back in x	
x = x * 10;	Multiply x by 10 and store back in x	
x = x /3;	divide x by 3 and store back in x	
x = x % 4;	makes x the remainder of x/4	

Operator	**Example Usage**	**Equivalent to**
+=	**x += 4;**	x = x + 4;
-=		x = x - 3;
***=**		x = x * 10;
/=		x = x /3;
%=		x = x % 4;

Example Usage	**Equivalent to**
x += 4 + b;	x = x + 4 + b;
x -= 3 * d	x = x – (3 * d);
x *= 10 - c	x = x * 10 - c ;
x /= b + c;	x = x / (b + c);
x %= d – c;	x = x % (d - c);

Precedence of combined operator is lower than regular operators.

Formatting output

cout object has a standard way of formatting variables. Also, it provides ways to format data as it is being displayed.

- Must include a preprocessor directive named **iomanip**; i.e., #include <iomanip>
- Common formatting that will be used in this class:
  - fixed
  - scientific
  - setprecision()
  - setw()
  - showpoint
  - left
  - right

Recall preprocessor directives learnt so far:

#include <iostream>	To use cin and cout objects
#include <string>	To use string objects.
#include <cmath>	To use mathematical functions, such as pow().
#include <iomanip>	To manipulate output.

Fixed Vs. Scientific

These are two different notations that you could choose from to display a number. Once set, the formatting will be set for the rest of the outputs until it is set to the other.

fixed: show the number with the fixed point notation. Usually followed by setprecision().

scientific: show the number in scientific notation. Usually followed by setprecision().

setprecision() Sets the precision of floating point number follows immediately after the decimal point. When less digits, than the precision, were in the number, zeros will be added to match up the precision. When more digts than the precision were in the number, last digit will be rounded up (if the following digits is 5 or above) or round down (if the following digit is less than 5).

fixed

e.g., no fraction part in the original number

```
        float floatNum = 2345678;

        cout << fixed;

        cout << setprecision(1) << floatNum << endl;        //2345678.0
        cout << setprecision(2) << floatNum << endl;        //2345678.00
        cout << setprecision(3) << floatNum << endl;        //2345678.000
        cout << setprecision(4) << floatNum << endl;        //2345678.0000
        cout << endl;
Output:
2345678.0
2345678.00
2345678.000
2345678.0000
```

e.g., fraction part is available in the original number

```
        float floatNum2 = 2345.678;

        cout << fixed;

        cout << setprecision(1) << floatNum2 << endl;        //2345.7
        cout << setprecision(2) << floatNum2 << endl;        //2345.68
        cout << setprecision(3) << floatNum2 << endl;        //2345.678
        cout << setprecision(4) << floatNum2 << endl;        //2345.6780
        cout << endl;
Output:
2345.7
2345.68
2345.678
2345.6780
```

When the system was set to fixed, the number is shown with no exponents. The precisions are counted from the original decimal point. If the original number has no decimal point or places, zero's will be appended after the decimal point.

Scientific

e.g., no fraction part in the original number

```
        float floatNum3 = 2345678;

        cout << scientific;

        cout << setprecision(1) << floatNum3 << endl;        //2.3e+6
        cout << setprecision(2) << floatNum3 << endl;        //2.35e+6
        cout << setprecision(3) << floatNum3 << endl;        //2.346e+6
        cout << setprecision(4) << floatNum3 << endl;        //2.3457e+6
        cout << endl;
Output:
2.3e+06
2.35e+06
2.346e+06
2.3457e+06
```

e.g., fraction part is available in the original number

```
        float floatNum4 = 2345.678;

        cout << scientific;

        cout << setprecision(1) << floatNum4 << endl;        //2.3e+3
        cout << setprecision(2) << floatNum4 << endl;        //2.35e+3
        cout << setprecision(3) << floatNum4 << endl;        //2.346e+3
        cout << setprecision(4) << floatNum4 << endl;        //2.3457e+3
        cout << endl;
Output:
2.3e+03
2.35e+03
2.346e+03
2.3457e+03
```

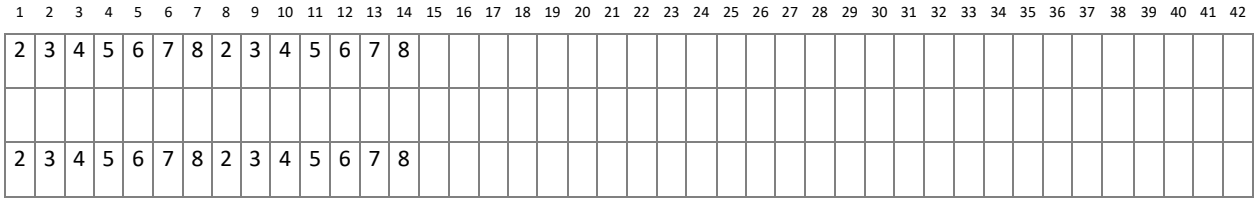
When the system was set to scientific, the number is shown in the scientific notation; i.e., with an exponent part. The precisions are counted from the decimal point of the scientific number.

setw(n)	Establishes a print field of n spaces (i.e., n spots)
left	causes subsequent output to be left justified. Usually followed by setw().
right	causes subsequent output to be right justified. Usually followed by setw().

Setw

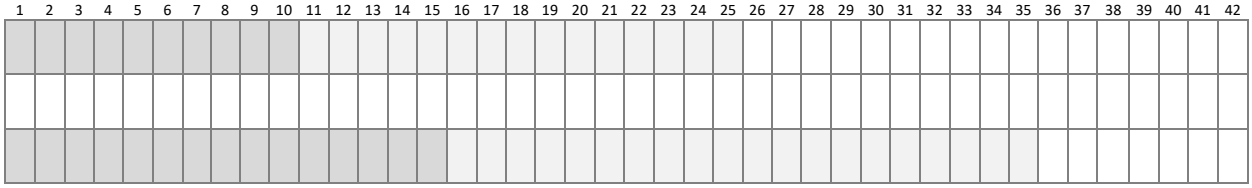
With NO setw()

```
        cout << fixed << left

        float floatNum = 2345678;
        cout << floatNum;
        cout << floatNum << endl;
        cout << endl;
        cout << floatNum;
        cout << floatNum << endl;
```



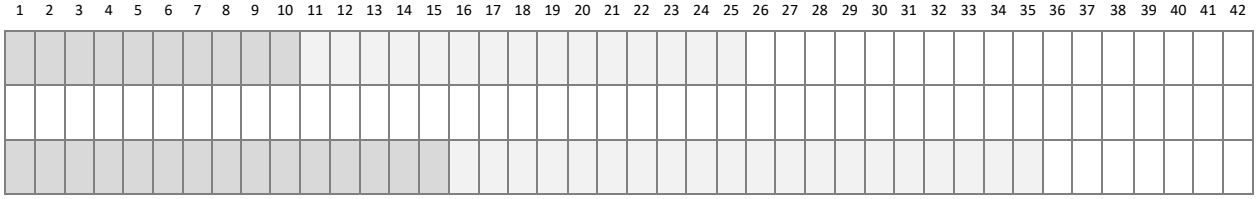
setw() with left aligned

```
        cout << fixed << left

        float floatNum = 2345678;
        cout << setw(10) << floatNum;
        cout << setw(15) << floatNum << endl;
        cout << endl;
        cout << setw(15) << floatNum;
        cout << setw(20) << floatNum << endl;
```



setw() with right aligned

```
        cout << fixed << right

        float floatNum = 2345678;
        cout << setw(10) << floatNum;
        cout << setw(15) << floatNum << endl;
        cout << endl;
        cout << setw(15) << floatNum;
        cout << setw(20) << floatNum << endl;
```



cin ignores leading white spaces and stops reading when it sees the first white space after the non-white character.

```
string fullName;
cout << "Enter your full name" << endl;
cin >> fullName;                                    //reads only one word


cout << endl << fullName ;
```

Output

Lasanthi Gamage [Enter]
Lasanthi

```
string fullName;
cout << "Enter your full name" << endl;
getline (cin,  fullName);                           //reads one (but entire) line


cout << endl << fullName ;
```

Output:

Lasanthi Gamage [Enter]
Lasanthi Gamage

```
char response;
cout << "Press any key to continue ....." << endl;
cin >> response;


cout << endl << "done!" ;                           //reads one non-white character
```

Output:

a [Enter]
done!

```
char response;
cout << "Press any key to continue ....." << endl;
cin.get(response);                                  // reads any one character

cout << endl << "done!" ;
```

Output:

Press any key to continue ….
[Enter]
done!

Mixing cin and get()

```
cout << "Enter   a number"  << endl;
cin >> number;
cout << "Enter any key to continue " << endl;
cin.get();
```

Output:

100[Enter]

Conclusion:

Does not allow to enter a character, because the keyboard buffer still have the [Enter] key.

Clearing the buffer

cin.ignore(n, c);	ignores n number of characters or until the character c is encountered.	e.g., cin.ignore(20, '\n');
cin.ignore();	Ignores the very next character.	

Summary

Statement	Notes	
cin >> letters;	letters can be a char or a string reads only one word (no white spaces are read, so the last white space is left)	Should use cin.ignore() AFTER the statement if a get function is used in the same program.
getline(cin, letters);	letters is a string. Reads the whole line.	Should use cin.ignore() BEFORE the statement if a cin statement is used in the same program.

cin.get(letters);
cin.get();
letters = cin.get();
	Letters is a char. Reads only one character (white or non)	

String member functions and operators:

```
string state = “Texas”;
int size = state.length();

concatenating

string firstName = “Lasanthi” ;
string lastName = “Gamge”;
string fullName = firstName + lastName;

cout << firstName << endl
       << lastName << endl
      << fullName << endl;
```

Output:

```
Lasanthi
Gamage
LasanthiGamage
```

Mathematical library functions

Need cmath library (as it was for pow()), i.e., #include <cmath>

Refer Table 3-13

y = abs(x)

e.g., both arguments and return value are in int.

```
 int magnitude;
magnitude = abs(-10);
```

y = sqrt(x)

e.g.,

```
double length ;
double area = 100.64;
length = sqrt(area);
```

arguments and return value are in doubles.

Random numbers

rand() - Use to generate a random number between 0 and a big number.

- Must include a preprocessor directive named cstdlib; i.e., #include < cstdlib >

```
int aNumber = rand();
```

issue: each time the program runs it would generate the same random number.

srand() - To get different random numbers, a seed number is provided to the generator.

This should be done before the rand() is called and only **once** for a program.

Must include the same preprocessor directive as for rand() cstdlib; i.e., #include < cstdlib >

```
short seed = 0;

srand(seed);
```

issue: If the seed is the same, the random numbers would be the same too.

time(0) – returns a number, number of seconds elapsed since mid night, 1970/01/01.

Must include a preprocessor directive named ctime; i.e., #include < ctime >

Pass this number as the seed for the srand() function.

```
short seed = time(0);
int randNum;

srand(seed);
randNum = rand();
cout << "randNum" << randNum;
```

Recall:

#include <iostream>	To use cin and cout objects
#include <string>	To use string objects.
#include <cmath>	To use mathematical functions, such as abs()
#include <iomanip>	To manipulate output.
#include <cstdlib>	for rand and srand
#include <ctime>	for the time function.

Example applications:

- Simulate an event of rolling a dice.
- Simulate an event of tossing a coin.
- Generate a password consists of 8 random characters.

Recall modular operator

10 % 3 = 1	13 % 3 = 1	11 % 4 = 3
11 % 3 = 2		12 % 4 = 0
12 % 3 = 0	10 % 4 = 2	

Generate a random number between 0 and 1. (i.e., two possible numbers)

Generate a random number between 0 and 2. (i.e., three possible numbers)

Generate a random number between 0 and 5. (i.e., 6 possible numbers)

Generate a random number between 2 and 7. (i.e., 6 possible numbers)

Generate a random number between MIN and MAX.

Generate a random DECIMAL number between 0 and 1: