CCN2042 Computer Programming Final Reminder

There may have mistakes because time is limited.
Examples in this reminder may cross over different topics, please be prepared before reading this reminder.
Note that the format and indention of output is incorrect because of Microsoft Word's formatting.

C++

High-level language and Object-oriented language

All data such as numbers, characters and strings are stored as a series of bits in a computer.
In the memory, number is stored as its binary format, character is stored based on ASCII encoding scheme.

6 phases in typical C++ environment

Creating – Preprocessing – Compiling – Linking – Loading – Execution

Appropriate indentation and spacing should be included for readability and easier debugging.
Syntax refers to the rules for creating a proper program.
Semantics refers to the meaning of a program.

ASCII

32 is ' '
65 is 'A'
97 is 'a', 65+32=97

Programming Errors

Syntax errors

- Refers to the mistake in a program that violates the program language rules.
- They can be detected during compilation.

Runtime errors

- Refers to the mistake in a program that causes the program to abort.
- i.e. The program terminates abnormally and cannot end successfully.

Logic errors

- Refers to the mistake in a program that produces incorrect results
- Also called bug.

CCN2042 Computer Programming Final Reminder

Variable Scope

A scope is a region of the program and broadly speaking there are three places, where variables can be declared:

- Inside a function or a block which is called local variables
- In the definition of function parameters which is called formal parameters
- Outside of all functions which is called global variables

Local Variables

Variables that are declared inside a function or block are local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own.

Global Variables

Global variables are defined outside of all the functions, usually on top of the program. The global variables will hold their value throughout the life-time of your program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration.

Declare/Initialize Variables

Declare variables

- Define a memory box for storing data

Initialize variables

- Give an initial value to the variable

Use ‘,’ to separate variables of the same type

lvalues and rvalues

Variables are lvalues and so they may appear on the left-hand side of an assignment. Numeric literals are rvalues and so they may not be assigned and cannot appear on the left-hand side.

lvalue

Expressions that refer to a memory location are called "lvalue" expressions. An lvalue may appear as either the left-hand or right-hand side of an assignment.

rvalue

The term rvalue refers to a data value that is stored at some address in memory. An rvalue is an expression that cannot have a value assigned to it which means an rvalue may appear on the right-hand side but not on the left-hand side of an assignment.

CCN2042 Computer Programming Final Reminder

Arithmetic Operators

+ - * / %

*integer division gives integer result

double b = 1 / 2 // b is 0, integer part of 0.5

Shortcut assignment operators

+= -= *= /= %=

Pre-increment ++x;

Post-increment x++;

Pre-decrement --x;

Post-decrement x--;

*Tips on counting number of digits of an integer

/10 getting all the digits except ones

%10 getting the digit ones

CCN2042 Computer Programming Final Reminder

Examples:

```
#include <iostream>
using namespace std;

int main()
{
	int num = 12345678, digit1[10] = {};
	int length = 0, digit2[10] = {};
	// Approach 1 pick out digit from right to left and put it into array
	// find length
	for (int x = num; x > 0; x /= 10)
	{
		length++;
	}
	int size = length;
	// put the right digit into array
	for (int x = num; x > 0; x /= 10)
	{
		digit1[--length] = x % 10;
	}
	// print array
	for (int i = 0; i < size; i++)
	{
		cout << digit1[i] << " ";
	}
	cout << endl;
	// Approach 2 pick out digit from left to right and put it into array
	// length is the denominator
	length = 1;
	for (int x = num; x > 10; x /= 10)
	{
		length*=10;
	}
	// put the left digit to array
	for (int i = 0; i<size;i++)
	{
		digit2[i] = num / length;
		num %= length;
		if (length >= 10)
			length /= 10;
	}
	// print array
	for (int i = 0; i < size; i++)
	{
		cout << digit2[i] << " ";
	}
	cout << endl;
	system("pause");
	return 0;
}
```

Output:

1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8

CCN2042 Computer Programming Final Reminder

Type Casting

Implicit:
Automatically performed during assignment of value

```
// E.g. 1 (implicit conversion)
float a = 12.8; int b;
b = a;  // float to int

cout << "a is " << a << endl;
cout << "b is " << b << endl;

// printout:
// a is 12.8
// b is 12
```

Explicit:
Specify the type when using a value

```
// E.g. 2 (explicit conversion)
int c = 65;

// int to char
cout << (char) c << " with ";
cout << "ASCII " << c << endl;

// printout:
// A with ASCII 65
```

Special Characters

\n \t \” \\

CCN2042 Computer Programming Final Reminder

Relational and Equality Operators

< <= > >= == !=

Logical Operators

! && ||

Boolean Variables

true(1 or NOT 0) or false(0)

if-else statement

```
if (bool)
{
        statements;
}
else if (bool)
{
        statements;
}
else
{
        statements;
}
```

switch statement

```
switch (either int or char)
{
case value1: statements; break;
case value2: statements; break;
case value3: statements; break;
default: statements;
}
```

**Notice that without break, statements of cases below are executed until break; is reached

Conditional Operator

(Boolean Expression) ? (true case) : (false case)

CCN2042 Computer Programming Final Reminder

Operator Precedence and Associativity

Operator precedence defines in one program statement with many operators, which operation should be considered and evaluated first.

Operator associativity defines in one program statement, among operators of the same precedence, how they are grouped in the absence of parentheses.

Operator	Associativity
()	Left to right
[] -> .	Left to right
++ -- (as postfix operators)	Right to left
++ -- (as prefix operators) ! + (unary) - (unary)	Right to left
~ & (address of) * (dereference)	Right to left
* / %	Left to right
+ -	Left to right
<< >>	Left to right
< <= > >=	Left to right
== !=	Left to right
&&	Left to right
||	Left to right
?:	Right to left
= += -= *= /= %=	Right to left

**Postfix > Prefix

Examples:

```
#include <iostream>
using namespace std;

int main()
{
	int x = 1, y = 2, z = 3;
	cout << x+++y+++z << endl;
	cout << x << " " << y << " " << z << endl;
	x = 1, y = 2, z = 3;
	cout << (x++)+(y++)+z << endl;
	cout << x << " " << y << " " << z << endl;
	system("pause");
	return 0;
}
```

Output:

6

2 3 3

6

2 3 3

CCN2042 Computer Programming Final Reminder

Repetition Statements

break;

- A break statement causes an immediate exit from the loop.

continue;

- A continue statement skips the remaining statements in the loop body, and proceeds with the next iteration of the loop.

while statement

```
while (continue loop condition)
{
        statements;
}
```

**Executing steps:

1. checking the “continue loop condition”.
2. if false, loop end.
3. otherwise, execute the statements and back to step 1.

for statement

```
for (initialize; continue loop condition; action after each loop)
{
        statements;
}
```

**Executing steps:

1. initialization
2. checking the “continue loop condition”.
3. if false, loop end.
4. otherwise, execute the statements.
5. execute “action after each loop” and back to step 2.

do-while statement

```
do {
        statements;
} while (continue loop condition);      // Remember the semi-colon “;”
```

**Executing steps:

1. execute the statements.
2. checking the “continue loop condition”.
3. if false, loop end.
4. otherwise, back to step 1.

CCN2042 Computer Programming Final Reminder

Nested loop

Steps to print pattern

1. add row-axis and col-axis to the pattern
2. modify the axis by observing the pattern sequence (ascending, descending)
3. first layer is looping about row
4. second layer is looping about column
5. use if-else statement to decide the content inside pattern (e.g. ‘ ’ or ‘*’)

```
for (about row)
{
        for (about col)
        {
                // if-else statement to print what you want
        }
cout << endl;
}
```

Examples: Floyd’s Triangle

```
#include <iostream>
using namespace std;

int main()
{
        int row,k=1;
        cout << "Enter number of rows: ";
        cin >> row;
        for (int i = 1; i <= row; i++) // rows
        {
                for (int j = 1; j <= i; j++) // cols
                        cout << k++ << " ";
                cout << endl;
        }
        system("pause");
        return 0;
}
```

Output:

```
Enter number of rows: 4
1
2 3
4 5 6
7 8 9 10
```

CCN2042 Computer Programming Final Reminder

Functions

Function is a set of program codes that performs a particular task. It can be from the C++ standard library or defined by the programmer.

Using functions in a program can facilitate the design, implementation, operation, and maintenance of large programs.

Functions allow programmers to modularize a program by separating its tasks into smaller, self-contained units.

Function prototypes and definition

Function prototypes are used as declaration of a function. For functions from the C++ standard library, their function prototypes are kept in the header file. Therefore, one needs to include the appropriate header files to use the functions from the C++ standard library.

User-defined functions are functions defined by the programmer (i.e., written by the programmer in the source file).

User-defined function needs to be declared before being used. A function can be declared by writing the function prototype or putting the function definition in the source code file in a place before the function is called.

Prototype:

```
functionType functionName (dataType);
```

Definition:

```
functionType functionName (datatype parameterName)
{
        body;
        return value; // if it is non-void
}
```

A reference variable is an alias (i.e. another name) for an already existing original variable. All operations performed on the reference variable are performed on the original variable.

In a function call, arguments can be passed into the function using either pass-by-value or pass-by-reference.

Pass-by-value

In pass-by-value, a copy of the argument's value is passed to the called function; changes to the copy in the function do not affect the variable's value in the caller.
In pass-by-reference, the called function can access and modify the caller's argument value directly.

Pass-by-reference

In pass-by-reference, the called function can access and modify the caller's argument value directly.

Pass-by-reference does not require copying of the argument's value to the parameter, and thus is able to avoid the overhead for copying. It is also useful for a function to "return" more than one value back to the caller.

Example:

```
#include <iostream>
using namespace std;

void printInt(int);
void addInt(int&,int);

int main() {
	int a = 1;
	printInt(a);
	addInt(a, 1);
	printInt(a);
	system("pause");
	return 0;
}

void printInt(int n /*Pass-by-value*/)
{
	cout << "Integer is " << n << endl;
}

void addInt(int& n /*Pass-by-reference*/, int x)
{
	n += x;
}
```

Output:
Integer is 1
Integer is 2

CCN2042 Computer Programming Final Reminder

Recursion

A recursive function is a function that calls itself directly or indirectly in its function body. Recursion is the scenario that a recursive function is called.

Examples: Fibonacci Series

```
#include <iostream>
using namespace std;

int fibonacci(int i)
{
	if (i == 0) // 0th
		return 0;

	if (i == 1) // 1st
		return 1;

	return fibonacci(i - 1) + fibonacci(i - 2); // sum of the two number in front
}

int main() {
	int i;

	for (i = 0; i < 10; i++) // showing from 0th to 9th
		cout << fibonacci(i) << " ";

	system("pause");
	return 0;
}
```

Output:

0 1 1 2 3 5 8 13 21 34

CCN2042 Computer Programming Final Reminder

Object-oriented Programming

Under object-oriented approach, the program is modelled by following the way people use to describe real-world objects.

Four basic elements in an object-oriented C++ program are: class, object, member function (i.e. the function definition inside a class), and data member (i.e. the variables declared for a class). Class members (member functions or data members) can be defined as public or private inside a class using the access-specifier labels.

Public and Private

Public class members can be accessed outside the class scope through the object; private class members can only be accessed inside the class scope (e.g. inside the member functions of the class).

Data hiding (or data encapsulation) refers to declaring data members of a class as private. It provides a better control on manipulating the data members of a class.

```
class className
{
public:
        // member functions
private
        // data member (no initialization is allowed)
};      // Remember the semi-colon ';'
```

Declaration: (inside class body – public)
fnType fnName (arguments);

Definition:
fnType className::fnName (arguments) {body}

CCN2042 Computer Programming Final Reminder

Constructer

A constructor of a class is used to initialize the object (its data members) when such object is created. It is a member function in the class with the same name as the class name and with no return type.

In a class definition without any constructor, the compiler provides a default constructor to the class implicitly. A default constructor is a constructor with no parameters (i.e. it takes no arguments).

Declaration: (inside class body – public)
className (arguments);

Definition:
className::className (arguments) {body}

Examples:

```
#include <iostream>
using namespace std;

class Time {
public:
	// Definition of a default constructor
	Time() {
		cout << "Default constructor is called." << endl;
	}
	// Definition of a constructor with parameters
		Time(int h, int m, int s) {
		cout << "Non-default constructor is called." << endl;
		hour = h;
		minute = m;
		second = s;
	}
	// To print the time values stored in the data members
	void printTime() {
		cout << "The time is ";
		cout << hour << ":";
		cout << minute << ":";
		cout << second;
		cout << endl;
	}
	void setTime(int h, int m, int s)
	{
		hour = h;
		minute = m;
		second = s;
	}
private:
	int hour;
	int minute;
	int second;
};

// program execution starts from the main function
int main() {
	// program statements to create two objects
	// default constructor call
	Time myTime1;
	// non-default constructor call
	Time myTime2(18, 40, 15);
	myTime1.setTime(15, 32, 25);
	cout << "Time of myTime1: ";
	myTime1.printTime();
	cout << "Time of myTime2: ";
	myTime2.printTime();
	return 0;
}
```

Output:

Default constructor is called.

Non-default constructor is called.

Time of myTime1: The time is 15:32:25

Time of myTime2: The time is 18:40:15

Memberwise assignment

objectA = objectB;

//all data mambers in object is assigned individually to same data members in object A

Examples: base on previous examples

```
// Only the main function is shown for brevity
int main() {
	Time myTime1;
	Time myTime2(18, 40, 15);

	// memberwise assignment
	myTime1 = myTime2;
	cout << "Time of myTime1: ";
	myTime1.printTime();
	cout << "Time of myTime2: ";
	myTime2.printTime();
	return 0;
}
```

Output:

Default constructor is called.

Non-default constructor is called.

Time of myTime1: The time is 18:40:15

Time of myTime2: The time is 18:40:15

CCN2042 Computer Programming Final Reminder

Arrays

Array is a data structure that contains related data items of the same type. An array occupies a consecutive group of memory locations when it is created.

An array index / subscript is the position number of a specific data (called element) in an array. It should be a non-negative integer and the first element has the index 0.

Size of an array remains the same once created.

It is important to avoid the off-by-one error and the array out-of-bound mistake—array index starts from 0, and is up to arraySize-1.

2-D array is a multidimensional array with two dimensions. In general, an array with m rows and n columns is called an m-by-n array.

An array name refers to the memory address location of the array.

When passing arrays into functions, the array is pass-by-reference by default.

A string is a series of characters treated as a single unit. It can include letters, digits, and special characters.

The cin statement reads characters from keyboard until the first whitespace character is encountered. Any characters typed after whitespace are stored in the array provided in the next cin.

In the C++ standard library, the <cstring> library provides string manipulation functions to work with string.

Declaration

dataType arrayName [numberOfElements];

Examples:

```
// Only the main function is shown for brevity
int main() {
	const int SIZE = 5; //SIZE is a constant integer
	Time t[5]; // 5 Time objects assume Time class has been defined
	double d[SIZE]; // SIZE number of elements
	char c[20];   // 20 character elements

	system("pause");
	return 0;
}
```

CCN2042 Computer Programming Final Reminder

Initialization

number of elements should be <= size of array while declare

If size of array hasn't specified while declaration, it will be same number as the number of elements through initialization.

for-loop to access the values of an array

```
for (int i = 0; i < SIZE; i++) {}
```

2D array

```
dataType arrayName [numberOfRows][numberOfCols];
```

Initialization

The initializer list can be nested

If nested,

- Each inner list specifies the values for the entire row, remaining elements in that row are filled with zeros
- Remaining rows in the outer list are filled with zeros

Else,

- Values are initialized from left to right, then top to bottom in the array
- Remaining elements are filled with zeros

nested-for-loop to access the values of an 2D array

```
for (int i = 0; i < numberOfRows; i++)
    for (int j = 0; j < numberOfCols; j++) {}
```

Passing arrays to function

Definition

```
fnType fnName (dataType paraName[], ...)   // 1D array
fnType fnName (dataType paraName[][numCols], ...)   // 2D array
```

Call

```
fnName( arrayName, ... );   // 1D array
fnName( arrayName[rowIndex], ... );   // 1D array parameter but arrayName is 2D array
                                      // Passing in a 1D array by specifying which row to pass
fnName( arrayName, ... );   // 2D array
```

Examples:

```
#include <iostream>
#include <iomanip>
using namespace std;
const int ROW = 3;
const int COL = 4;
void print1DArray(int a[], int size) {
	for (int i = 0; i < size; i++) {
		cout << setw(3) << a[i] << " ";
	}
	cout << endl;
}
void print2DArray(int a[][COL], int size) {
	for (int i = 0; i < size; i++) {
		for (int j = 0; j < COL; j++) {
			cout << setw(3) << a[i][j] << " ";
		}
		cout << endl;
	}
}
int main() {
	int b[COL] = { 1, 2, 3, 4 };
	int c[ROW][COL] = { { 11, 12, 13, 14 },
	{ 15, 16, 17, 18 },
	{ 19, 20, 21, 22 } };
	cout << "Contents of array b: " << endl;
	print1DArray(b, COL);
	cout << "Contents of array c: " << endl;
	print2DArray(c, ROW);
	cout << "Contents of array c[1]: " << endl;
	print1DArray(c[1], COL);
	return 0;
}
```

Output:

Contents of array b:

1 2 3 4

Contents of array c:

11 12 13 14

15 16 17 18

19 20 21 22

Contents of array c[1]:

15 16 17 18

CCN2042 Computer Programming Final Reminder

Using character array to form a string

Declaring and initializing

- Using initializer list
  char arrayName[] = { val1, val2, …, valN, '\0' };
- Using string constant
  char arrayName[] = stringConstant;
  stringConstant should be enclosed in double-quotes ""

cin and cout

cin >> arrayName;
cout << arrayName;

Examples:

```
#include <iostream>
using namespace std;

int main() {
	char s[20];
	cout << "Enter a string (without space): ";
	cin >> s;
	cout << "The input string is:" << endl;
	cout << s;
	cout << endl;
	return 0;
}
```

Output:
Enter a string (without space): HelloWorld
The input string is:
HelloWorld

for-loop to access specific characters in a character array

for (int i = 0; array[i] != '\0'; i++) {}

CCN2042 Computer Programming Final Reminder

Pointers

A pointer variable contains memory address as values. When a pointer "points to" a variable, the pointer variable is storing the memory address of that variable.

A pointer can be used to indirectly reference the variable that it points to.

The address-of operator (&) and the dereferencing operator (*) are inverse of each other.

Arrays and pointers are closely related. One can set a pointer to point to an array, and use the pointer to refer to the array element.

By passing the address of a variable or an array into a function that receives pointer argument, pass-by-reference happens and the function can modify the value of the variable or the array.

Principle of least privilege refers to awarding a function just enough access to the information and resources to accomplish a task, but no more.

To follow the principle of least privilege when designing programs, one has to choose the appropriate way to pass pointers to function:

(1) using non-constant pointer to non-constant data;
(2) using non-constant pointer to constant data;
(3) using constant pointer to nonconstant data;
(4) using constant pointer to constant data.

Pointer-based string is a series of characters which is pointed to by a pointer to character.

When a pointer-based string is printed, the first character to print is one which the pointer is pointing to, until a null character is encountered.

When a character pointer is initialized with string literal, a read-only memory would be reserved for holding the pointer-based string.

An array of pointer-based strings can be used to manipulate multiple strings. The array stores pointers to character, and each pointer points to the first character of one string.

CCN2042 Computer Programming Final Reminder

& and *

dataType * ptrName = address;
ptrName = newAddress;
* ptrName = newValue

Examples:

```
#include <iostream>
using namespace std;

int main() {
	int a = 5;
	int * ptr = &a;
	int * aPtr;
	cout << "Address of a: " << &a << endl;
	aPtr = ptr;
	cout << "Value of aPtr: " << aPtr << endl;
	cout << "Value of *aPtr: " << *aPtr << endl;
	cout << "Updating *aPtr to 10..." << endl;
	*aPtr = 10;		// use *aPtr as lvalue
				// indirectly modify variable a
	cout << "New value of a: " << a << endl;
	cout << "New value of *aPtr: " << *aPtr << endl;
	return 0;
}
```

Output:
Address of a: 012FFDD4
Value of aPtr: 012FFDD4
Value of *aPtr: 5
Updating *aPtr to 10...
New value of a: 10
New value of *aPtr: 10

CCN2042 Computer Programming Final Reminder

Setting a pointer to point to an array

ptrName = arrayName;

ptrName = &arrayName[0];

Pointer offset notation

*(ptrName + i) // dereference the location ptrName + i

Examples:

```
#include <iostream>
using namespace std;

int main() {
	int v[5] = { 2,4,6,8,10 };
	int * vPtr;
	vPtr = v; // or vPtr = &v[0];
		  // use pointer to access different array elements by offset notation
		  // Print v[1]
	cout << "*(vPtr + 1) is " << *(vPtr + 1) << endl;
	// Modify v[3] to 0
	*(vPtr + 3) = 0; // dereference the location vPtr + 3
			 // change the content to 0
	cout << "*(vPtr + 3) is " << *(vPtr + 3) << endl;
	return 0;
}
```

Output:

*(vPtr + 1) is 4

*(vPtr + 3) is 0

Passing pointer to function

fnType fnName (dataType * paraName, …) {}

Calling a function with pointer parameter – always pass the address

1. Pass the address of a variable using &
2. Pass the address of an array using arrayName

Examples:

```
#include <iostream>
using namespace std;
// Simulates pass-by-reference behaviour
void squareByPointer(int * a) {
	*a = *a * *a; // Deference the pointer to get and
	              // change the content
}
int main() {
	int num = 5, v[1] = { 5 };
	cout << "Before function call, num = " << num << endl;
	squareByPointer(&num); // pass address into the function
	cout << "After function call, num = " << num << endl;
	cout << "Before function call, v[0] = " << v[0] << endl;
	squareByPointer(v); // pass address into the function
	cout << "After function call, v[0] = " << v[0] << endl;
	return 0;
}
```

Output:

Before function call, num = 5

After function call, num = 25

Before function call, v[0] = 5

After function call, v[0] = 25

CCN2042 Computer Programming Final Reminder

Using const with pointers

Non-constant pointer to non-constant data
dataType * ptrName

- Data can be modified
- Pointer can be modified

Non-constant pointer to constant data
const dataType * ptrName

- Data cannot be modified
- Pointer can be modified

Constant pointer to non-constant data
dataType * const ptrName

- Data can be modified
- Pointer cannot be modified

Constant pointer to constant data
const dataType * const ptrName

- Data cannot be modified
- Pointer cannot be modified

Pointer-based string

Declaring and initializing

1. Points to a string constant (read-only)
   char * ptrName = strConst;
2. Points to the memory reserved for character array
   char arrayName [numOfChars];
   char * ptrName = arrayName;
3. Points to some new memory
   char * ptrName = new char [numOfChars];
   delete[] ptrName // after use

cin cout of pointer-based string

Examples:

```
#include <iostream>
using namespace std;

int main() {
	char a[50];
	char * s1 = "Hello World";
	char * s2 = a;
	char * s3 = new char[50];
	// cin >> s1; // runtime error
	// because s1 points to read only memory
	cout << "Original s1: " << s1 << endl;
	s1 = s1 + 3; // move s1 to 3 characters ahead
	cout << "New s1: " << s1 << endl;
	cout << "Input for s2: "; cin >> s2;
	cout << "Input for s3: "; cin >> s3;
	cout << "s2: " << s2 << endl;
	cout << "s3: " << s3 << endl;
	return 0;
}
```

Output:

Original s1: Hello World

New s1: lo World

Input for s2: Computer

Input for s3: Programming

s2: Computer

s3: Programming

Arrays of pointers to characters

char * arrayName[numOfPtrs] = { strConst1, strConst2, …};

Examples:

```
#include <iostream>
using namespace std;

int main() {
	char * brands[4] = { "Apple", "Samsung", "Microsoft", "Google" };
	cout << "Printing the brands one by one:" << endl;
	for (int i = 0; i < 4; i++)
	{
		cout << brands[i] << endl;
	}
	return 0;
}
```

Output:

Printing the brands one by one:

Apple

Samsung

Microsoft

Google

Arrow member selection operator

Examples:

```
#include <iostream>
using namespace std;
class Time {
public:
	void printTime() {
		cout << "The time is 15:42" << endl;
	}
};
int main() {
	Time t1;
	Time * tPtr = &t1;
	t1.printTime(); // Use the dot (.) to call a member function
	tPtr->printTime(); // Use the arrow (->) to call a member function
			   // Reason: tPtr is a pointer to Time class
	(*tPtr).printTime(); // convert pointer back to object
	return 0;
}
```

Output:

The time is 15:42

The time is 15:42

The time is 15:42

#include <iostream>
using namespace std;

Common header

cin.getline(arrayName, size, delimiter);

- Copy input into specified arrayName until either
  1. size – 1 is reached
  2. the delimiter character is input

Examples:

```
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
	char s[20];
	cout << "Enter a string: ";
	cin.getline(s, 20, '\n'); // if cin >> s is used,
				  // only the first word is read in
	cout << "The input string is:" << endl;
	cout << s;
	cout << endl;
	return 0;
}
```

Output:
Enter a string: Hello World
The input string is:
Hello World

sizeof(dataType)

- Returns the size of type

  Examples:

```
int a;
double b;
char c;
int arr[10];
int * aPtr;
cout << "Size of an integer: " << sizeof(int) << endl;
cout << "Size of a double: " << sizeof(double) << endl;
cout << "Size of character c: " << sizeof c << endl;
cout << "Size of array arr: " << sizeof arr << endl;
cout << "Size of an integer pointer aPtr: " << sizeof aPtr << endl;
```

  Output:
  Size of an integer: 4
  Size of a double: 8
  Size of character c: 1
  Size of array arr: 40
  Size of an integer pointer aPtr: 4

CCN2042 Computer Programming Final Reminder

#include <iomanip>

Required header file for stream manipulators

Non-sticky manipulator

- Produce a setting that applies for the next output value only

Sticky manipulator

- Produces a setting that remains until the setting is changed in the program

endl

- To output a new line

setw(n)

- To set the field width as n; field width means the width (the number of character positions) the output occupies (non-sticky)

left / right

- To left or right justify the output in a field (sticky)

fixed / scientific

- To print floating point numbers using fixed-point formatting / scientific formatting (sticky) scientific

$$X.XXXXXXe \pm XX$$

Precision value

- The precision value defines how precise a floating point value is printed. In default format, it refers to the maximum number of meaningful digit to print; in fixed-point format or scientific format, it refers to the number of decimal places, with trailing zeros, to print.

setprecision(n)

- To change the system precision value (default is 6) (sticky)

Examples:

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
	int a = 6, b = 9.6, c = 10;
	char d = 'N';
	double x, y, z;
	x = (a + b) / 2;
	y = (x + c) / 3;
	z = d + c / 3 + y;

	cout << x << " " << y << " " << z << endl;
	cout << fixed;
	cout << x << " " << y << " " << z << endl;
	cout << scientific;
	cout << x << " " << y << " " << z << endl;
	cout << setprecision(4);
	cout << x << " " << y << " " << z << endl;

	system("pause");
	return 0;
}
```

Output:

7 5.66667 86.6667

7.000000 5.666667 86.666667

7.000000e+000 5.666667e+000 8.666667e+001

7.0000e+000 5.6667e+000 8.6667e+001

CCN2042 Computer Programming Final Reminder

#include <ctime>

Required header file for time() function
time_t time (time_t* timer);
calling time(0) return current time

#include <cstdlib>

Required header file for rand() and srand()

void srand (unsigned int seed);
usually use current time as seed, i.e. srand(time(0));

int rand (void)
calling rand() return a pseudo-random integral number

SKILL:
General formula

$$[shifter] + rand(\,)\%\left(\frac{[maximum]}{[multiplier]} + 1\right) * [multiplier]$$

$[shifter] = \ range\ between\ final\ minimum\ and\ 0$
$[maximum] = original\ maximum - [shifter]$
$[multiplier] = special\ requirements$
ODD number, $[multiplier] = 2$
EVEN number, $[multiplier] = 2$
Multiples of n, $[multiplier] = n$

Steps constructing random statement:
Using Example 1 (random odd integer from 5 to 127 inclusively)

1. find out real range
   i.e. [5,7,9,11,13,…,119,121,123,125,127]
   i.e. minimum = 5 and original maximum = 127
2. calculate [shifter] by shifting the minimum to 0
   i.e. [shifter] = 5
3. calculate [maximum] by minus the original maximum by [shifter]
   i.e. [maximum] = 127 – 5 = 122
4. calculate [multiplier]
   since odd, [multiplier] = 2
5. build general formula, i.e. $5 + rand(\,)\%62 * 2$
6. check again using first two and last two of the rand()%x
   i.e. [0,1,60,61]
   x2 [0,2,120,122]
   +5 [5,7,125,127]
7. shifter may need to +1 or -1 if odd or even number is required

Examples:
random odd integer from 5 to 127 inclusively

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main() {
	srand(time(0));
	for (int x = 0; x < 10; x++)
	{
		cout << 5 + rand() % ((127 - 5) / 2 + 1) * 2 /*Answer*/ << ' ';
	}
	cout << endl << endl;
	// Demo of all integer between 0 and (127 - 5) / 2 + 1 = 62 (without 62)
	// i.e. the range of rand() % 62 = [0,62)
	for (int x = 0, y = 1; x < (127 - 5) / 2 + 1; x++, y++)
	{
		cout << x << '\t';
		if (y % 10 == 0)
		{
			cout << endl;
			y = 0;
		}
	}
	cout << endl << endl;
	// Demo of all EVEN integer between 0 and ((127 - 5) / 2 + 1) * 2 = 124
	(without 124)
	// the range of rand() % 62 * 2 = [0,124) and EVEN integer
	for (int x = 0, y = 1; x < ((127 - 5) / 2 + 1) * 2; x += 2, y++)
	{
		cout << x << '\t';
		if (y % 10 == 0)
		{
			cout << endl;
			y = 0;
		}
	}
	// Demo of all ODD integer between 0 + 5 = 5 and 5 + ((127 - 5) / 2 + 1) * 2 =
	129 (without 129)
	// the range of 5 + rand() % 62 * 2 = [0,129) and ODD integer
	cout << endl << endl;
	for (int x = 0, y = 1; x < ((127 - 5) / 2 + 1) * 2; x += 2, y++)
	{
		cout << x + 5 << '\t';
		if (y % 10 == 0)
		{
			cout << endl;
			y = 0;
		}
	}
	cout << endl << endl;
	system("pause");
	return 0;
}
```

Output:
55 27 67 105 119 89 107 7 35 127

0 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61

0 2 4 6 8 10 12 14 16 18
20 22 24 26 28 30 32 34 36 38
40 42 44 46 48 50 52 54 56 58
60 62 64 66 68 70 72 74 76 78
80 82 84 86 88 90 92 94 96 98
100 102 104 106 108 110 112 114 116 118
120 122

5 7 9 11 13 15 17 19 21 23
25 27 29 31 33 35 37 39 41 43
45 47 49 51 53 55 57 59 61 63
65 67 69 71 73 75 77 79 81 83
85 87 89 91 93 95 97 99 101 103
105 107 109 111 113 115 117 119 121 123
125 127

random multiples of 8 from -10 to 89 inclusively

**Notice that real range is [-8,0,8…72,80,88]

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main() {
	srand(time(0));
	for (int x = 0; x < 10; x++)
	{
		cout << -8 + rand() % ((88 - -8) / 8 + 1) * 8 /*Answer*/ << ' ';
	}
	cout << endl << endl;
	// Demo of all integer between 0 and (88 - -8) / 8 + 1 = 13 (without 13)
	// i.e. the range of rand() % 13 = [0,13)
	for (int x = 0, y = 1; x < (89 - -8) / 8 + 1; x++, y++)
	{
		cout << x << '\t';
		if (y % 10 == 0)
		{
			cout << endl;
			y = 0;
		}
	}
	cout << endl << endl;
	// Demo of all 8 multiples between 0 and ((88 - -8) / 8 + 1) * 8 = 104 (without
	104)
	// the range of rand() % 13 * 8 = [0,104) and multiples of 8
	for (int x = 0, y = 1; x < ((88 - -8) / 8 + 1) * 8; x += 8, y++)
	{
		cout << x << '\t';
		if (y % 10 == 0)
		{
			cout << endl;
			y = 0;
		}
	}
	// Demo of all 8 multiples between 0 - 8 = -8 and -8 + ((88 - -8) / 8 + 1) * 8
	= 96 (without 94)
	// the range of -8 + rand() % 13 * 8 = [-8,96) and multiples of 8
	cout << endl << endl;
	for (int x = 0, y = 1; x < ((88 - -8) / 8 + 1) * 8; x += 8, y++)
	{
		cout << x - 8 << '\t';
		if (y % 10 == 0)
		{
			cout << endl;
			y = 0;
		}
	}
	cout << endl << endl;
	system("pause");
	return 0;
}
```

Output:
8 0 0 32 48 32 40 80 8 -8

0 1 2 3 4 5 6 7 8 9
10 11 12

0 8 16 24 32 40 48 56 64 72
80 88 96

-8 0 8 16 24 32 40 48 56 64
72 80 88

#include <cstring>

strcpy(array1, array2)

- Copy second argument into first argument

  Examples:

```
char s1[] = "abcd";
char s2[5];
strcpy(s2, s1);
```

  Result:

  s2 contains “abcd”

strncpy(array1, array2, n)

- Copy n characters (third argument) from second argument into first argument

  Examples:

```
char s1[] = "abcd";
char s2[5];
strncpy(s2, s1, 3);
s2[3] = '\0';
```

  Result:

  s2 contains “abc”

  **Note: s2 needs to be null-terminated manually

strcat(array1, array2)

- Append second argument to first argument

  Examples:

```
char s1[10] = "abcd";
char s2[10] = "efg";
strcat(s1, s2);
```

  Result:

  s1 contains “abcdefg”

  s2 does not change

strncat(array1, array2, n)

- Append n characters (third argument) from second argument into first argument

  Examples:

```
char s1[10] = "abcd";
char s2[10] = "efg";
strncat(s1, s2, 2);
```

  Result:

  s1 contains “abcdef”

  s2 does not change

strcmp(array1, array2)

- Compare between first and second arguments character-by-character

  Examples:

```
char s1[] = "abcd";
char s2[] = "abdd";
cout << strcmp(s1, s2);
```

  Result:

  -1 is printed

strncmp(array1, array2, n)

- Compare between first and second arguments character-by-character by n characters (third argument)

  Examples:

```
char s1[10] = "abcd";
char s2[10] = "abdd";
cout << strncmp(s1, s2, 2);
```

  Result:

  0 is printed

strlen(array)

- Return the number of characters in the argument

  Examples:

```
char s1[10] = "abcd";
cout << strlen(s1);
```

  Result:

  4 is printed

#include <cmath>

Function	Header File	Purpose	Parameter(s) Type	Result
abs(x)	<cmath>	Returns the absolute value of its argument: **abs(−7) = 7**	int (double)	int (double)
ceil(x)	<cmath>	Returns the smallest whole number that is not less than **x : ceil(56.34) = 57.0**	double	double
cos(x)	<cmath>	Returns the cosine of angle: **x : cos(0.0) = 1.0**	double (radians)	double
exp(x)	<cmath>	Returns e^x, where **e = 2.718 : exp(1.0) = 2.71828**	double	double
fabs(x)	<cmath>	Returns the absolute value of its argument: **fabs(−5.67) = 5.67**	double	double
floor(x)	<cmath>	Returns the largest whole number that is not greater than **x : floor(45.67) = 45.00**	double	double
pow(x, y)	<cmath>	Returns x^y; if x is negative, y must be a whole number: **pow(0.16, 0.5) = 0.4**	double	double
sqrt(x)	<cmath>	Returns the nonnegative square root of x; x must be nonnegative: **sqrt(4.0) = 2.0**	double	double

Miscellaneous, Tips and Skills

Execution Flow

Executing a statement from left to right step by step, you can only move on to next part after the first part is totally finished.

Examples:

```
#include <iostream>
using namespace std;

bool fn(int x)
{
	cout << "Calling fn(" << x << ")\n";
	if (x < 0)
		return false;
	if (fn(x - 1))
		return true;
	return fn(x-1) && fn(x-2);
}

int main() {
	if (fn(2))
		cout << "YES\n";
	else
		cout << "NO\n";
	return 0;
}
```

Output:

Calling fn(2)
Calling fn(1) // checking 2nd if of fn(2)
Calling fn(0) // checking 2nd if of fn(1)
Calling fn(-1) // checking 2nd if of fn(0) and return false
Calling fn(-1) // fn(0) running last return, and fn(-1) return false
Calling fn(0) // fn(1) running last return and calling fn(0)
Calling fn(-1) // checking 2nd if of fn(0) and return false
Calling fn(-1) // fn(0) running last return, and fn(-1) return false
Calling fn(1) // fn(2) running last return and calling fn(1)
Calling fn(0) // checking 2nd if of fn(1)
Calling fn(-1) // checking 2nd if of fn(0) and return false
Calling fn(-1) // fn(0) running last return, and fn(-1) return false
Calling fn(0) // fn(1) running last return and calling fn(0)
Calling fn(-1) // checking 2nd if of fn(0) and return false
Calling fn(-1) // fn(0) running last return, and fn(-1) return false
NO // because fn(2) return false

**Note that the computer will not execute the 2nd part of last return as it is the relationship of && and the 1st part of the return statement is false, i.e. the last statement is same as return false

CCN2042 Computer Programming Final Reminder

The const statements

The complier can check any writing access is made to a given variable

Examples:

```
#include <iostream>
using namespace std;

int main() {
	const int x = 15;
	bool cond = (x = 13);
	return 0;
}
```

Output:

ERROR, ‘x’: cannot assign to a variable that is const

```
#include <iostream>
using namespace std;

int main() {
	const int x = 15;
	int *p = &x;
	return 0;
}
```

Output:

ERROR, ‘initializing’: from ‘const int *’ to ‘int *’ discards qualifiers

The comma operator

A succession of expressions separated by commas are evaluated from left to right, the results of the global expression being the value of the last one to be evaluated:

Examples:

```
#include <iostream>
using namespace std;

int main() {
	const int x = 15;
	int i = 0, j = 0, k = 0;
	cout << (i++, j += 14, k -= 3) << endl;
	cout << i << " " << j << " " << k;
	return 0;
}
```

Output:

-3

1 14 -3

CCN2042 Computer Programming Final Reminder

General Bugs and Mistakes

The program never stops

Examples:

```
for (int i = 1; i > 0; i++)
	cout << i << " ";
for (int j = 0; j < 100; j *= 2)
	cout << j << " ";
for (int k = 10000; k < 10; k / 10)
	cout << k << " ";
```

201718 Semester One PP

The code below are just a sample, there are many approach of coding can get the same result.

B1

(a)

EXECUTE YOUR OWN

(b)

```
cout << "Input an integer: ";
cin >> n;
cout << fixed << setprecision(5);
if (n == 0)
	cout << "\"ZERO\" is not accepted\n";
else
	cout << "Reciprocal = " << 1.0 / n << endl;
```

B2

(a)

EXECUTE YOUR OWN

(b)(i)

```
int c = 0;
for (a; a <= b; a++)
{
	cout << setw(5) << a;
	if (++c % 6 == 0)
		cout << endl;
}
```

(b)(ii)

```
for (int i = a; i < b / 2; i++)
{
	if (a%i == 0 && b%i == 0)
		gcd = i;
}
```

B3

(a)

EXECUTE YOUR OWN

(b)

```
int sumR(int n, int& rcount)
{
	int sum;
	if (n / 10 == 0)
	{
		rcount = 1;
		return n;
	}
	sum = n % 10 + sumR(n / 10, rcount);
	rcount++;
	return sum;
}
```

B4

(a)

```
private:
	double x, y;
```

(b)

```
Vector(int a, int b)
{
	x = a;
	y = b;
}
```

(c)

```
double magnitude()
{
	return sqrt(x*x + y * y);
}
```

(d)

```
double angle()
{
	return atan((double)y / x) / PI * 180;
}
```

(e)

EXECUTE YOUR OWN

B5

(a)

```
for (int i = n; i > 0; i /= 10)
	length++;
for (int i = n, j = length; i > 0; i /= 10, j--)
	digits[j - 1] = i % 10;
```

(b)

```
for (int i = 0; i < size; i++)
{
	cout << data[i] << "|";
	for (int j = 0; j < data[i]; j++)
	{
		cout << "*";
	}
	cout << endl;
}
```

(c)

EXECUTE YOUR OWN

B6

(a)

EXECUTE YOUR OWN

(b)

```
for (int i = 0; i < NUM_STRINGS; i++)
	for (int j = 0; *(str[i] + j) != '\0'; j++)
		if (((int)*(str[i] + j) - 104) >= 0
			&& ((int)*(str[i] + j) - 104) < LETTERS)
			freq[(int)*(str[i] + j) - 104]++;
```

(c)

```
cout << "Letter Occurence:\n";
for (int i = 0; i < LETTERS; i++)
	cout << (char)(i + 104) << ": " << freq[i] << endl;
```

CCN2042 Computer Programming Final Reminder

C1

(a)

```
int input;
cout << "Input the secret message:\n";
cin.getline(m, MAX_SIZE, '\n');
cout << "Input key (1-9): ";
cin >> input;
while (input < 1 || input > 9)
{
	cout << "Out-of-range. Input key (1-9): ";
	cin >> input;
}
key = input;
```

(b)

```
for (int i = key; m[i] != '\0'; i += key)
{
	char a = (char)(33 + rand() % 94);
	char temp1 = m[i];
	m[i] = a;
	for (int j = i + 1; ; j++)
	{
		char temp2 = m[j];
		m[j] = temp1;
		temp1 = temp2;
		if (m[j] == '\0')
		{
			char temp2 = m[j];
			m[j] = temp1;
			temp1 = temp2;
			break;
		}
	}
	i++;
}
```

(c)

```
for (int i = 0; m[i] != '\0'; i++)
	if ((int)m[i] >= 65 && (int)m[i] <= 90)
		m[i] = ((int)m[i] + key <= 90) ? m[i] + key : m[i] + key - 25;
	else if ((int)m[i] >= 97 && (int)m[i] <= 122)
		m[i] = ((int)m[i] + key <= 122) ? m[i] + key : m[i] + key - 25;
```

(d)

```
char temp[MAX_SIZE] = {};
for (int i = 0, j = 0, k = 1; ; i++, j += key)
{
	if (m[i] == '\0')
	{
		temp[i] = m[i];
		break;
	}
	if (j >= strlen(m))
		j = k++;
	temp[i] = m[j];
}
for (int i = 0; i < strlen(temp); i++)
	m[i] = temp[i];
```

(e)

```
char temp[MAX_SIZE] = {     };
for (int i = 0;; i++)
{
	temp[i] = m[i];
	if (m[i] == '\0')
		break;
}
for (int i = 0, j = 0, k = 1; temp[i] != '\0'; i++, j += key)
{
	if (j >= strlen(temp))
		j = k++;
	m[j] = temp[i];
}
for (int i = 0; m[i] != '\0'; i++)
	if ((int)m[i] >= 65 && (int)m[i] <= 90)
		m[i] = ((int)m[i] - key >= 65) ? m[i] - key : m[i] - key + 25;
	else if ((int)m[i] >= 97 && (int)m[i] <= 122)
		m[i] = ((int)m[i] - key >= 97) ? m[i] - key : m[i] - key + 25;
for (int i = key; i < strlen(m); i += key)
	for (int j = i;m[j-1]!='\0'; j++)
		m[j] = m[j + 1];
```

C2

(a)

```
cout << s << endl;
cout << "row? ";
cin >> r;
cout << "col? ";
cin >> c;
while (c < 1 || c > COL || r < 1 || r > ROW)
{
	cout << "Input error, try again.\n";
	cout << "row? ";
	cin >> r;
	cout << "col? ";
	cin >> c;
}
```

(b)

```
for (int i = 0; i < row; i++)
	for (int j = 0; j < col; j++)
		a[i][j] = 1 + rand() % 9;
```

(c)

```
cout << s << endl;
for (int i = 0; i < row; i++)
{
	for (int j = 0; j < col; j++)
		cout << setw(4) << a[i][j];
	cout << endl;
}
```

(d)

```
for (int i = 0; i < row; i++)
	for (int j = 0; j < col; j++)
		c[i][j] = a[i][j] + b[i][j];
```

(e)

```
(row > col) ? col = row : row = col;
for (int i = 0; i < row; i++)
	for (int j = 0; j < col; j++)
		b[i][j] = a[j][i];
```

(f)

```
for (int i = 1; i <= row; i++)
{
	cout << "Input " << col << " numbers for row " << i << ":\n? ";
	for (int j = 0; j < col; j++)
		cin >> a[i - 1][j];
}
```

(g)

```
if (c1 != r2)
	return false;
else
	for (int i = 0; i < r1; i++)
		for (int j = 0; j < c2; j++)
			for (int m = 0, n = 0; m < r2, n < c1; m++, n++)
				c[i][j] += a[i][m] * b[n][j];
```

CCN2042 Computer Programming Final Reminder

201617 Semester 2 PP

C1

(a)

```
void separate(double num, int places, int&integer, int & decimal)
{
	integer = num;
	int temp = 1;
	for (int i = 0; i < places; i++)
		temp *= 10;
	decimal = (num - integer)*temp;
}
```

(b)

```
cout << "   Integer   Decimal\n";
cout << setw(10) << integer << setw(10) << decimal << endl;
```

C2

(a)

```
for (int i = 2; i < x; i++)
	if (x%i == 0)
		return false;
return true;
```

(b)

```
do {
	for (int i = 2; i <= n; i++)
	{
		if (isPrime(i) && n%i == 0)
		{
			cout << i << " ";
			n /= i;
			break;
		}
	}
} while (n > 1);
```

C3

```
maxLength = 0;
for (int i = 0; i < NUM_WORDS; i++)
	for (int j = 1; words[i][j-1] != '\0'; j++)
		if (maxLength < j)
			maxLength = j;
cout << "The longest word has " << maxLength << " characters\n";
for (int i = 0; i < maxLength; i++)
{
	for (int j = 0; j < NUM_WORDS; j++)
	{
		if (i >= strlen(words[j]))
		{
			cout << "  ";
			continue;
		}
		cout << words[j][i] << " ";
	}
	cout << endl;
}
```

C4

```
class MyString
{
public:
	MyString(const char s[])
	{
		int i = 0;
		do{
			content[i] = s[i];
		} while (s[i++]!='\0');
		length = --i;
	}
	void print()
	{
		cout << content << endl;
	}
	int getLength()
	{
		return length;
	}
	int indexOf(char c)
	{
		int i = 0;
		for (i; i < length; i++)
			if (content[i] == c)
				return i;
		return -1;
	}
private:
	char content[SIZE];
	int length;
};
```