Chapter 3 Variables and Operators

Input, process, output…input, process, output…input, process, output… 输入；处理；输出

It's all computers do. Think about it: no matter what software you're using, that's all that's going on. Input, process, output (IPO), is a computer model that all processes in a computer must follow.

For example:

- Input e.g. read from network/disk/database/hardware, accept user input.
- Process e.g. FFT, sum, product, random shuffle.
- Output e.g. write to network/disk/database, flash lights, change display, respond to user input.

Before we can do any processing, we need to have data. And once we get that data we need to hold onto it in some way. How do we do this? We use variables. 变量

3.1 Variables

A variable is a "container" in which a data value can be stored inside the computer's memory. 值 内存

The stored value can be referenced by using its name later in the program. 引用

A variable looks like a bottle in somewhat. A bottle can be used to hold different liquids such as water, a variable can hold or store the value of different data type.

Let's imagine that if you were asked to remember a number: 5. What happened in your brain? You stored this value in your memory. Then, if you were asked to add 6 to the number, you should be retaining the numbers 11 (that is 5+6) in your memory.

The whole process described above is a simile of what a computer can do with two variables. That can be expressed in C/C++ with the following set of statements:

```
a = 5;
b = 6;
c = a + b;
```

Variables are the lifeblood of software—the medium through which data travels all around your programs. The operations described in this chapter demonstrate how to store, process, assign, manipulate and transfer data through the use of variables.

血液；媒介

赋值；操作；传输

3.2 Variables and identifiers

标识（zhi）符

Each variable needs a name that identifies it and distinguishes it from the others. In the following chapters, we will give name to other programming elements, such as functions, classes, etc. All these elements have a common name—identifier.

An identifier is a name that is assigned by the user for a program element such as variable, type, template, class, function or namespace.

赋予，给予

When naming an identifier, we should obey the naming convention. A naming convention is a set of rules for choosing the character sequence to be used for identifiers.

命名约定

(1) Only alphabets, digits and underscores are permitted.
(2) It must begin with either a letter or an underscore.
(3) Key words cannot be used as a name.
(4) Upper case and lower case letters are distinct. C/C++ is case-sensitive.

字母；数字；下画线

关键字

区分大小写的

3.3 Data Types

数据类型

Each variable has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory.

特定的类型

There are following basic types of variable in C/C++ in Table 3-1.

基本类型

Table 3-1 Data Types in C/C++

Type	Description	Size
bool	Stores either value true or false	1 Byte
char	Storage of individual text	1 Byte
int	Storage of numbers without a fractional part	4 Bytes
float	A single-precision floating point value	4 Bytes
double	A double-precision floating point value	8 Bytes

布尔型
字符型
整型
浮点型
双精度型

Please notes:

- The character type "char" can be used to store numbers, but is generally used to store characters, e.g. 'a', 'H', '?'. The character is stored as a number which specified by ASCII.
- The range of values that can be stored in the int type is limited +2147483647. And although this seems to be a very large number, the population of China is currently (2020) 1366000000. So you would not be able to use an int to store the total savings of the population of China, for example.
- The float type allows a wide range of values to be stored, but there are limits in precision, that is how accurately a number can be represented in a floating point format. A float will be able to accurately represent a number to 9 significant places.
- The bool type is used when looking at logical expressions where there are only two possible results True and False.

字符型

ASCII 码

格式
有效数字位数
逻辑表达式

The data types described above cover nearly all of the basic data types in C/C++. However, there is always an exception and in C/C++ this is void. The void data type represents nothing and this will make little sense to you until we discuss functions .

空类型

函数

3.4 Variables Declaration and Initialization

变量的声明和初始化

A variable must be declared before it is used.

A declaration specifies a type, and contains a list of one or more variables of that type as follows:

确定

variableType variablelList;

Here, “variableType” must be a valid C/C++ data type including char, int, float, double, bool or any user defined object, etc, and “VariableList” may consist of one or more identifier names separated by commas. Some valid declarations are shown here:

合法的

用逗号分开

```
int i, j, k;
char c, ch;
float f, salary;
double d;
```

Variables are initialized (assigned a value) with an equal sign followed by a constant expression. The general form of initialization is:

等号

常数表达式

variable_name = value;

Note that the use of the ‘=’ sign is for assignment rather than equivalence. Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows:

type variable_name = value;

For examples: int number = 12;

Let’s look at this line of C/C++ code in more detail:

int: this tells the compiler what the type of the variable is. The compiler needs to know about the type of the variable because the amount of memory used will be different and the way that the pattern of individual binary digits (bits) which is decoded into a number will be different.

二进制数字（位）

number: the name or label for the variable. Once memory has been allocated to store our integer value, our programme can access that memory using the variable name.

标签；分配

访问

=: This is an assignment operation that stores the number 12 in our variable.

赋值运算

When C++ declares a variable, a block of the computer's memory will be used to store the value, it may already contain a bit pattern (from operations performed elsewhere in your program or another program running on the computer). This means that when we create a new variable, it will contain a random value. Therefore, it is important that you assign a value to a variable before you start to use it.

计算机内存块
位（二进制）格式
随机值

The program is below(see Figure 3-1).

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

int main()
{
    int a = 3, b = 5;        // initializes a and b.
    float  c = 22.5;         // initializes c.
    double pi = 3.14159;     // declares an approximation of pi.
    char x = 'x';            // the variable x has the value 'x'.

    return 0;
}
```

Figure 3-1 Assignment for different variables

3.5 Variable Names and Comments

变量名和注释

Often in some C/C++ textbooks or when you look at coding examples on the web, you see code that looks like:

```
int a, b, c;
float d, e;
```

This is valid C/C++ code, the first line declares three variables of type int with the names or labels of a, b and c. The second line declares two variables to hold floating point data and gives them the names d and e.

Although this code will work, it is not recommended when writing C/C++ programs. It is a legacy of the times when computers were very limited in terms of memory storage and computing power. Today, the mobile phone in your pocket or a laptop computer on your desk have computing power more than a mainframe computer that would have been shared between 20 to 30 users at the year when C was developed.

过去的；过时的
大型主机

Modern C/C++ programming practice will use longer names for variables. The name given to each variable should show what the variable will store.

If we have a variable called a, what does it store? Is it the number of apples in a shop or the number of pigs that a farmer owns or the telephone number of a friend? The variable name tells the person reading the program nothing useful. This means that we would have to add comments to our code. Comments are text in our source file that is ignored by our compiler.

程序注释

A single line comment is indicated by using the symbol "//". The compiler will ignore all characters after the "//" symbol until the end of the line. Our code with comments might look like:

单行注释

```
int a;   //used to store the number of apples in shop
int b;   //used to store the number of pigs on a farm
int c;   //used to store current telephone number
```

Even with comments we have to remember what each variable in the program does, and we may need to keep looking back at the comments so that we are reminded of what the variable holds. The preferred technique in modern coding is to use what is known as self-commenting. In this approach the name of a variable indicates what it is stored in the variable. Using this approach, we would declare our variables as:

技术

自我注释

```
int applesInShop;
int pigsOnFarm;
int currentPhoneNumber;
```

You can see, we hope, that the variable names show the information that we expect the variable to hold.

Styles of Variable Names

变量命名风格

C/C++ does not allow us to have space characters in a variable name. So we need to be able to tell where the individual words in the variable name start. Some people use underscore characters "_" where the spaces are:

```
apples in shop becomes apples_in_shop
```

In this book, we are using "camel case" where we don't use underscore

驼峰式命名方式

characters to represent spaces, but we use a capital or upper case letter at the start of each word in the variable name.

```
apples in shop becomes applesInShop
```

You can also see that the variable name starts with a small or lower case letter. We will always follow this style throughout this book and you should do the same.

Unlike some scripting languages, in C/C++ you must formally declare a variable, that is, to define its type and name before we can use it elsewhere in our program. This strict typing approach means that we have to think a little more when we are writing our code but are less likely to have our code failing to work correctly when it is executed.

脚本语言；正式声明一个变量

C/C++ allows you to declare variables without assigning values to them in which case the variable will have a random value, this can cause your program to perform incorrectly or crash. It is good practice to give a variable a value when you create it, and avoiding the circumstances such as dividing by zero, multiplying by infinity, etc.

赋值
随机值
崩溃
乘以一个无穷大的数

If you want to store a different value to your variable, then you need to assign a new value to the variable using the assignment operator "=", the general form for assignment is:

```
variableName = variableValue;
```

The value of a variable can be changed by setting a literal value, or by assigning another variable to it, in which case the variable just takes on the value of the variable on the right, that is to say, the original value of the variable is overridden.

文本（值）
被覆盖了

```
newBunnies = 15;
bunnyCounter = newBunnies;
```

3.6 Operators

运算符

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.

操作

C++ is rich in built-in operators and provides the following types of operators:

内置的，内建的

- Arithmetic operators 算术运算符
- Relational operators 关系运算符
- Logical operators 逻辑运算符
- Bitwise operators 位运算符
- Assignment operators 赋值运算符

This chapter will examine the arithmetic, relational, logical, bitwise, assignment operators one by one.

3.6.1 Arithmetic Operators

As a engineer, much of the data processing you are most likely to perform is going to be arithmetic based. Most programming languages are geared around arithmetic, with C/C++ being no exception.

数据处理

There are following arithmetic operators supported by C/ C++ language in Table 3-2.

Table 3-2 Arithmetic operators (Assume variable A holds 10 and variable B holds 20)

Operator	Description	Example
+	Addition, adds two operands	A+B will give 30
−	Subtraction, subtracts second operand from the first	A−B will give −10
*	Multiplication, multiplies both operands	A*B will give 200
/	Division	B/A will give 2
%	Modulus operator, remainder of after an integer division	B%A will give 0
+ +	Increment operator, increases integer value by one	A+ + will give 11
− −	Decrement operator, decreases integer value by one	A− − will give 9

运算数

模运算；余数

自增运算符

自减运算符

We often find in the code that we want to add something to the contents of a variable and store the results in the original variable. The following line of code

will perform this task.

```
applesInShop += applesFromFarm;
```

The operator "+=" takes the contents of the variable applesFromFarm and adds it to the contents of the variable applesInShop where the result of the operation is stored. It is equivalent to the statement below:

```
applesInShop = applesInShop +applesFromFarm;
```

There is a minor advantage in terms of the use of memory by using this operator.

小的优点（内存的占用小一些）

There are corresponding operators for subtraction, multiplication and division.

Table 3-3 Arithmetic operators

Operation	Original values		Value stored in variable **a** after operation
	a	b	
a += b;	4	2	6
a −= b;	4	2	2
a *= b;	4	2	8
a /= b;	4	2	2

3.6.2 Logical Operators

逻辑运算符

There are three logical operators supported by C/C++ language: logical AND, logical OR and logical NOT.

Table 3-4 gives a explaination for each logical operator.

Table 3-4 Logical operators (Assume variable A holds 1 and variable B holds 0)

Operator	Description	Example
&&	Logical AND. If both the operands are true, then condition becomes true	(A&&B) is false
||	Logical OR. If any of the two operands is true, then condition becomes true	(A||B) is true
!	Logical NOT. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false	! A is false ! B is true

3.6.3 Bitwise Operators

位运算符

位与位操作

A bitwise operator works on bits and perform bit-by-bit operation. The truth table for &, |, and ^ are as follows:

Table 3-5 Bitwise operators in C/C++

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

The Bitwise operators supported by C/C++ language are listed in Figure 3-2. Assume variable A holds 60 and variable B holds 13, then:

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

main()
{
   unsigned int a = 60;    // 60 = 0011 1100
   unsigned int b = 13;    // 13 = 0000 1101
   int c = 0;

   c = a & b;              // 12 = 0000 1100
   cout << "Line 1 - Value of c is : " << c << endl;
   c = a | b;              // 61 = 0011 1101
   cout << "Line 2 - Value of c is: " << c << endl;
   c = a ^ b;              // 49 = 0011 0001
   cout << "Line 3 - Value of c is: " << c << endl;
   c = ~a;                 // -61 = 1100 0011
   cout << "Line 4 - Value of c is: " << c << endl;
   c = a << 2;             // 240 = 1111 0000
   cout << "Line 5 - Value of c is: " << c << endl;
   c = a >> 2;             // 15 = 0000 1111
   cout << "Line 6 - Value of c is: " << c << endl;
   return 0;
}
```

Figure 3-2 Bitwise operators example

The output of the program above is as Figure 3-3.

```
C:\MyProject\MyFirstProject\bin\Debug\MyFirstProject.exe
Line 1 - Value of c is : 12
Line 2 - Value of c is: 61
Line 3 - Value of c is: 49
Line 4 - Value of c is: -61
Line 5 - Value of c is: 240
Line 6 - Value of c is: 15

Process returned 0 (0x0)   execution time : 0.978 s
Press any key to continue.
```

Figure 3-3 Output of the bitwise operators example

3.6.4 Relational Operators

关系运算符

There are following relational operators supported by C/C++ language in Table 3-6.

Table 3-6 Relational operators in C/C++ (Assume variable A holds 10 and variable B holds 20)

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes, then condition becomes true	(A==B) is not true
!=	Checks if the values of two operands are equal or not, if values are not equal, then condition becomes true	(A!= B) is true
>	Checks if the value of left operand is greater than the value of right operand, if yes, then condition becomes true	(A>B) is not true
<	Checks if the value of left operand is less than the value of right operand, if yes, then condition becomes true	(A<B) is true
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes, then condition becomes true	(A>=B) is not true
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes, then condition becomes true	(A<=B) is true

3.6.5 Operators Precedence in C/C++

运算符优先级

Operator precedence determines the value of an expression. Certain operators have higher precedence than others. For example, the multiplication operator has

higher precedence than the addition operator. For example:

```
x = 7 + 3 * 2;
```

Here, x is assigned 13, not 20, because operator * has higher precedence than.

In Table 3-7, operators with the highest precedence appear at the top, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Table 3-7 Operators Precedence in C/C++

Category	Operator	Associativity	
Postfix	() [] -> . ++ − −	Left to right	后置（运算符）
Unary	+ − ! ~ ++ − − (type)* & sizeof()	Right to left	一元运算符
Multiplicative	* / %	Left to right	乘、除、求余
Additive	+ −	Left to right	加、减
Shift	<< >>	Left to right	移位
Relational	< <= > >=	Left to right	关系运算
Equality	== !=	Left to right	相等性
Bitwise AND	&	Left to right	位与
Bitwise XOR	^	Left to right	位异或
Bitwise OR	|	Left to right	位或
Logical AND	&&	Left to right	逻辑与
Logical OR	||	Left to right	逻辑或
Conditional	?:	Right to left	条件运算
Assignment	= += −= *= /= %=>>= <<= &= ^= |=	Right to left	赋值
Comma	,	Left to right	逗号运算符

Below (see Figure 3-4) is the example for demonstrating the precedence of operators.

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

main() {
   int a = 20;
   int b = 10;
   int c = 15;
   int d = 5;
   int e;

   e = (a+b)*c/d;       // (30*15)/5
   cout << "Value of (a+b)*c/d is:" << e << endl;
   e = ((a+b)*c)/d;     // (30*15)/5
   cout << "Value of ((a+b)*c)/d is:" << e << endl;
   e = (a+b)*(c/d);     // (30)*(15/5)
   cout << "Value of (a+b)*(c/d) is:" << e << endl;
   e = a+(b*c)/d;       //  20+(150/5)
   cout << "Value of a+(b*c)/d is:" << e << endl;
   return 0;
}
```

Figure 3-4 Example of the precedence of operators

The output is as below (see Figure 3-5):

```
C:\MyProject\MyFirstProject\bin\Debug\MyFirstProject.exe
Value of (a+b)*c/d is:90
Value of ((a+b)*c)/d is:90
Value of (a+b)*(c/d) is:90
Value of a+(b*c)/d is:50

Process returned 0 (0x0)   execution time : 0.488 s
Press any key to continue.
```

Figure 3-5 Output for code of precedence of operators

3.7 Some Much Used Operators in C / C++

3.7.1 Increment and Decrement Operator

自增和自减运算符

The increment operator ++ adds 1 to its operand, and the decrement operator— subtracts 1 from its operand. Both the increment and decrement operators can either precede (prefix) or follow (postfix) the operand.

前置的；后置的

When an increment or decrement is used as part of an expression, there is an important difference in prefix and postfix forms.

> Increment or decrement will be done before rest of the expression for a prefix increment or decrement operation, and increment or decrement will be done after the complete expression is evaluated for a postfix increment or decrement operation.

Let's see some examples of ++ as prefix and postfix in C and C++ (see Figure 3-6).

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

int main()
{
   int var1 = 1, var2 = 1;

   cout << var1++ << endl;  // var1 is displayed as 1, then it is increased to 2.
   cout << var1++ << endl;


   cout << ++var2 << endl;  // var2 is increased to 2 then, it is displayed.

   return 0;
}
```

Figure 3-6 Increment and decrement operators

3.7.2 sizeof() Operator

The sizeof() operator is a compiling time unary operator which can be used to compute the size of its operand. The result of sizeof() is of unsigned integral type. sizeof can be applied to any data-type, including primitive types such as integer and floating-point types, pointer types, or compound datatypes such as classes, structures, unions and any other user defined data type. The syntax of using sizeof is as follows:

一元运算符
无符号的整数类型（的数）
指针类型的；混合类型，比如类、结构体、联合体（都是数据类型）

```
sizeof (data type)
```

The following examples demonstrate the sizeof operator available in C and C++ (see Figure 3-7).

```
int main()
{
   cout << "Size of char : " << sizeof(char) << endl;
   cout << "Size of int : " << sizeof(int) << endl;
   cout << "Size of float : " << sizeof(float) << endl;
   cout << "Size of double : " << sizeof(double) << endl;
   cout << "Size of bool : " << sizeof(bool) << endl;

   return 0;
}
```

Figure 3-7 Demonstration of the sizeof operator

The output is:

```
Size of char : 1
```

```
Size of int : 4
Size of float : 4
Size of double : 8
Size of bool : 1
```

Note: sizeof() may give different output according to machines.

3.7.3 Modulus (%) Operator

模运算符（求余）

The modulus operator produces the remainder of an integer division.

余数

Syntax: If x and y are integers, then the expression:

```
x % y
```

produces the remainder when x is divided by y.

> The % operator cannot be applied to floating-point numbers, i.e float or double. If you try to use the modulus operator with floating-point constants or variables, the compiler will produce an error.

- If x is completely divided by y, the result of the expression is 0.
- If x is not completely divisible by y, then the result will be the remainder in the range [1, y−1].
- If y is 0, then a compile-time error will be produced.

整除

Basic code for explaination is as below (see Figure 3-8).

```
#include <stdio.h>
int main()
{
    int x, y, z;
    x = 6;
    y = 5;
    z = 3;

    printf("%d\n", x % y);
    printf("%d\n", y % x);
    printf("%d\n", x % z);
    printf("%d\n", z % x);

    return 0;
}
```

Figure 3-8 Demonstration of modulus operator

3.7.4 Conditional Operator (? :)

条件运算符

```
variable = expression1 ? expression2 : expression3;
```

Where expression1, expression2, and expression3 are expressions, variable holds the value of the entire expression.

First, expression1 is evaluated, if it is true, then expression2 is evaluated and becomes the value of the entire expression. If expression1 is false, then expression3 is evaluated and its value becomes the value of the expression.

The conditional operator is a kind of similar to the if-else statement that we will learn in the next chapter. It follows the same algorithm as of if-else statement but the conditional operator takes less space and helps to write the if-else statements in the shortest way. It can be visualized into if-else statement as:

```
if(expression1==true)
{
   variable = expression2;
}
else
{
   variable = expression3;
}
```

Since the conditional operator takes three operands to work, hence it is also called ternary operator, and it is the only ternary operators in C and C++.

三元运算符

Below is the example of conditional operator.

```
#include<iostream>
using namespace std;
int main()
{
   int x = 3, y = 5, bigNumber;
   bigNumber = (x > y ) ? x : y;
   cout << "The big number is: " << bigNumber << endl;
   return 0;
}
```

3.7.5 comma “,” operator

逗号运算符

In a C/C++ program, comma is used in two contexts: (1) A separator (2) An

Operator. For example:

```
#include<iostream>
using namespace std;
int main()
{
   int a = 3, b = 4, c = 5;    //commas are used as separators
   cout << "a=" << a << endl;
   cout << "b=" << b << endl;
   cout << "c=" << c << endl;
   return 0;
}
```

In the example above, commas work as separators, not as operators. The first line of statement declares three variables and then do assignment one by one from left to right. Below is the example for comma operator used as an operator:

```
#include<iostream>
using namespace std;
int main()
{
   int a;
   a = 1, 2, 3;
   cout<< a << endl;
   return 0;
}
```

In the program above, comma works as an operator. The comma operator has the lowest precedence of any operator, so the assignment operator takes precedence over comma and the expression “a = 1, 2, 3” becomes equivalent to “(a = 1), 2, 3”. The output is 1.

最低的运算优先级

For the program segment below:

程序段

```
int a=1, b=2, c=3, d;
d = (a, b, c);
```

Commas act as separators in the first line and as an operator in the second line.

In the second line, the round brackets are used, so comma operator is executed first. The comma expression (a, b, c) is a sequence of expressions which

圆括号

顺序（执行）表达式

evaluates to the last variable c, so the value of d is 3.

Look at the program segments below:

```
int a=3, b=4, c=5, d;
d = (a += 1, a + b, a + c);
```

For the line 2 of statement, the expression (a += 1, a + b, a + c) is separated into three parts by two commas:

The first part increases value of a by 1, so a holds 4.

The value of the second part is 8 (4+4).

The value of the third part is 9 (4+5).

As the value of the entire expression (a += 1, a + b, a + c) is the value of third part, so the value of d is 9.

Chapter Review

1. What is a variable?

2. Evaluate the following expressions:

   a. 12*3 + 4
   b. 5+3/2
   c. 4/5*5
   d. 11%3
   e. (4+5)/3

Programming Exercises

1. Write a program in C to print the sum of two numbers, and then change this program into a C++ language program.

2. Write a program in C++ to find the size of fundamental data types using the sizeof() operator. The output should look like this:

```
Find Size of fundamental data types :
----------------------------------------
The sizeof char is: 1 bytes.
The sizeof int is: 4 bytes.
The sizeof float is: 4 bytes.
The sizeof double is: 8 bytes.
The sizeof bool is: 1 bytes.
```

3. Assume an integer takes 4 bytes, what is the output of the program below?

```
#include<stdio.h>
int main()
{
   int i = 5, j = 10, k = 15;
   printf("%d ", sizeof(k /= i + j));
   printf("%d", k);
   return 0;
}
```

(A) 4 1

(B) 4 15

(C) 2 1

(D) Compile-time error

4. What is the output of the program below?

```
#include<stdio.h>
int main()
{
   int i = (1, 2, 3);
   printf("%d", i);
   return 0;
}
```

(A) 1

(B) 3

(C) Garbage value

(D) Compile time error

5. What is the output of the program below?

```
#include<stdio.h>
int main()
{
   int a = 1;
   int b = 1;
   int c = a || --b;
   int d = a-- && --b;
   printf("a = %d, b = %d, c = %d, d = %d", a, b, c, d);
   return 0;
}
```

(A) a = 0, b = 1, c = 1, d = 0

(B) a = 0, b = 0, c = 1, d = 0

(C) a = 1, b = 1, c = 1, d = 1

(D) a = 0, b = 0, c = 0, d = 0

6. Write a program to produce the output as shown below:

```
Results:
x value    y value    expressions   results
10         | 5        | x+=y       | x=15
10         | 5        | x-=y-2     | x=7
10         | 5        | x*=y*5     | x=250
10         | 5        | x/=x/y     | x=5
10         | 5        | x%=y       | x=0
```

7. Write a program in C++ to find the area and perimeter of a rectangle.

Sample output is:

```
Find the Area and Perimeter of a Rectangle :
-------------------------------------------
Please input the length of the rectangle : 3
Please input the width of the rectangle : 4
The area of the rectangle is : 12
The perimeter of the rectangle is : 14
```