Loops and Decisions

Ho Dac Hung

Relational Operators

- A relational operator compares two values. The values can be any built-in C++ data type, such as char, int, and float, or—as we'll see later—they can be user-defined classes. The comparison nvolves such relationships as equal to, less than, and greater than. The result of the comparison is true or false; for example, either two values are equal (true), or they're not (false).

Relational Operators

```
int main()
{
    int numb;
    cout << “Enter a number: “;
    cin >> numb;
    cout << “numb<10 is “ << (numb < 10) << endl;
    cout << “numb>10 is “ << (numb > 10) << endl;
    cout << “numb==10 is “ << (numb == 10) << endl;
    return 0;
}
```

Relational Operators

>	Greater than
<	Less than
==	Equal to
!=	Not equal to
>=	Greater than or equal to
<=	Less than or equal to

Relational Operators

jane = 44; //assignment statement

harry = 12; //assignment statement

(jane == harry) //false

(harry <= 12) //true

(jane > harry) //true

(jane >= 44) //true

(harry != 12) // false

(7 < harry) //true

(0) //false (by definition)

(44) //true (since it’s not 0)

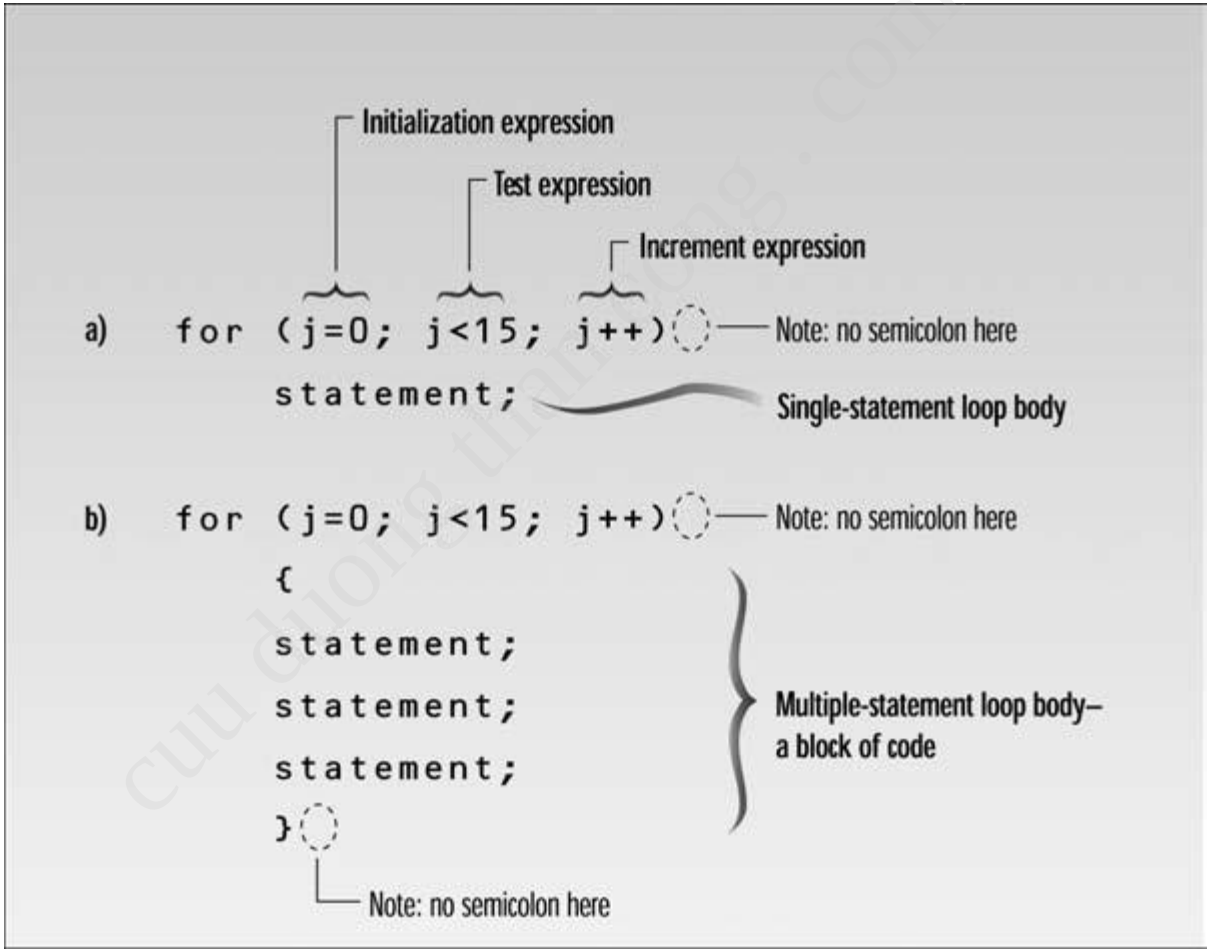
Loops

- Loops cause a section of your program to be repeated a certain number of times. The repetition continues while a condition is true. When the condition becomes false, the loop ends and control passes to the statements following the loop.

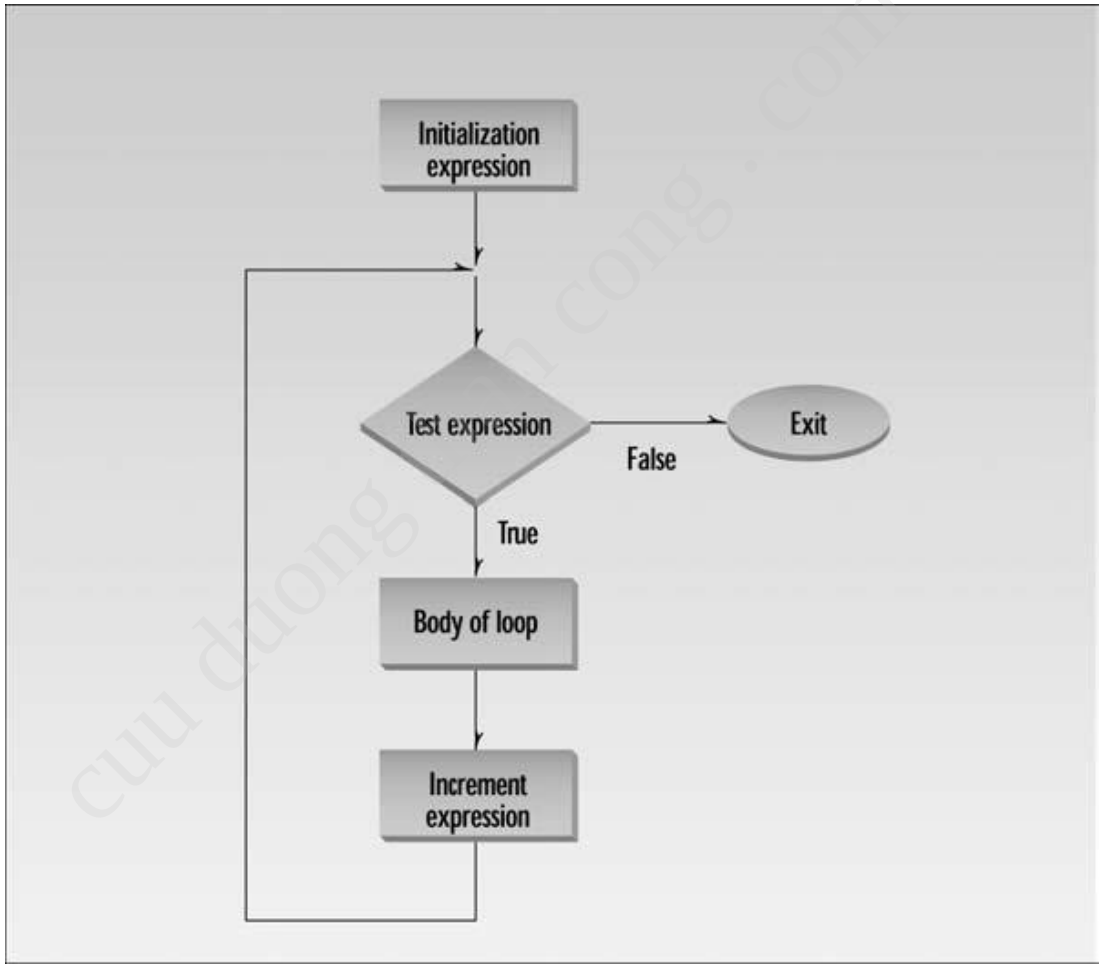
The for Loop

- The for loop executes a section of code a fixed number of times. It's usually (although not always) used when you know, before entering the loop, how many times you want to execute the code.

The for Loop



The for Loop



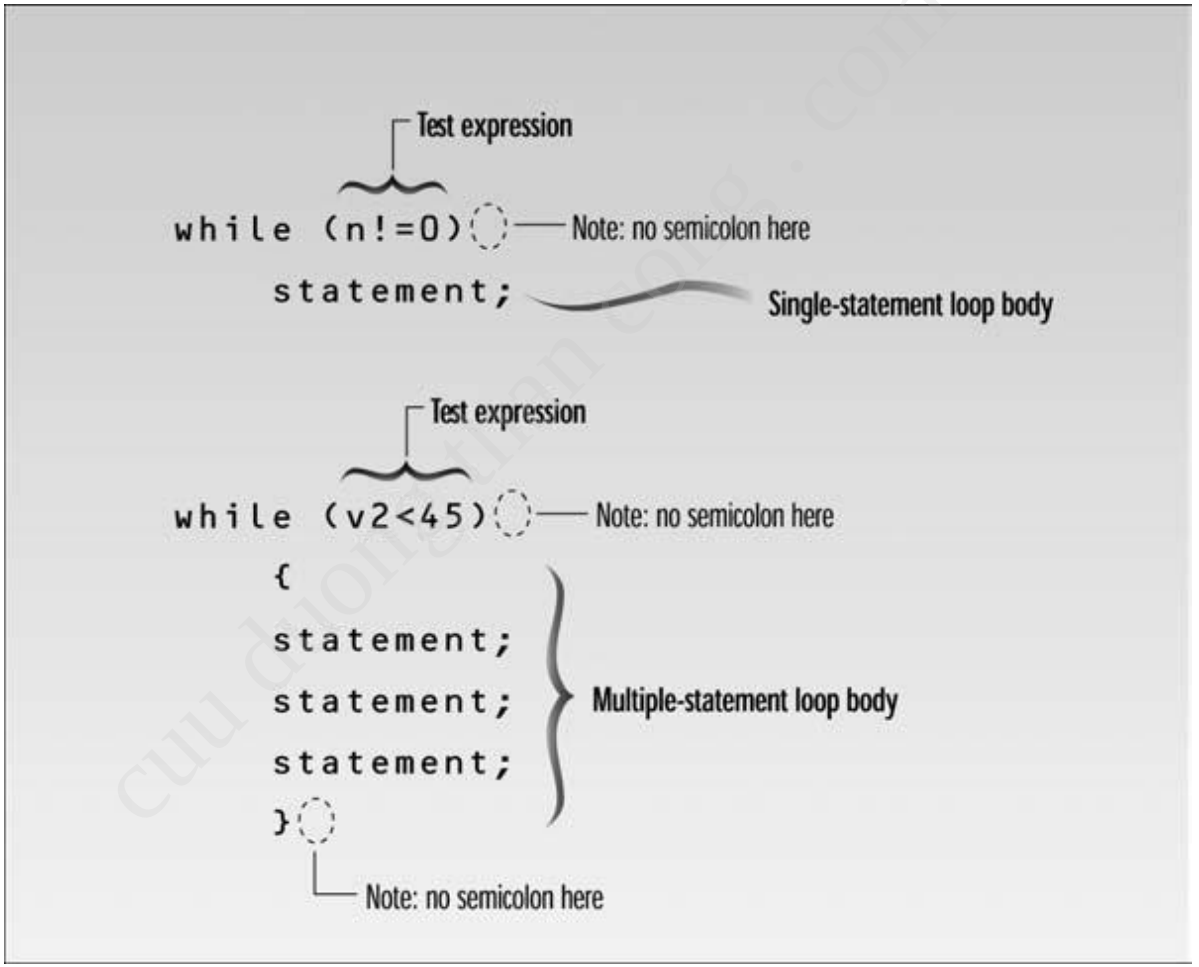
The for Loop

```
for(numb=1; numb<=10; numb++)
{
    cout << setw(4) << numb;
    int cube = numb*numb*numb;
    cout << setw(6) << cube << endl;
}
```

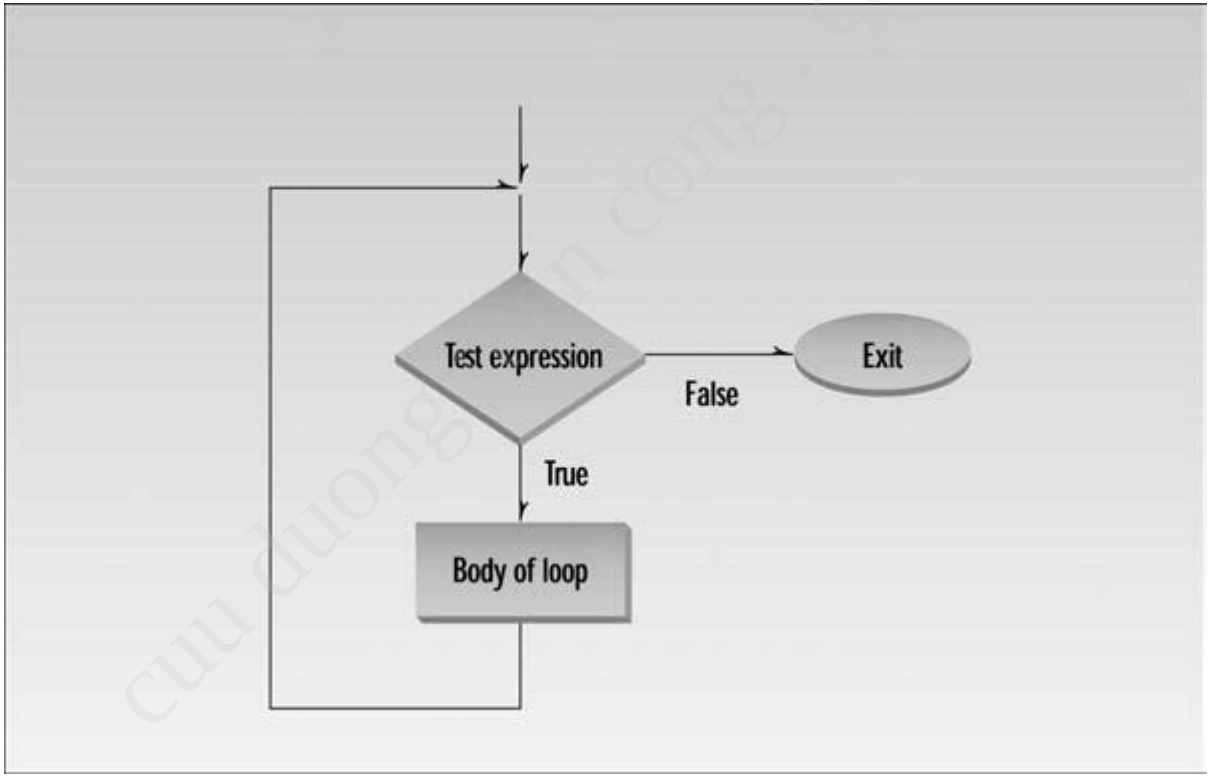
The while Loop

- The for loop does something a fixed number of times. What happens if you don’t know how many times you want to do something before you start the loop? In this case a different kind of loop may be used: the while loop.

The while Loop



The while Loop



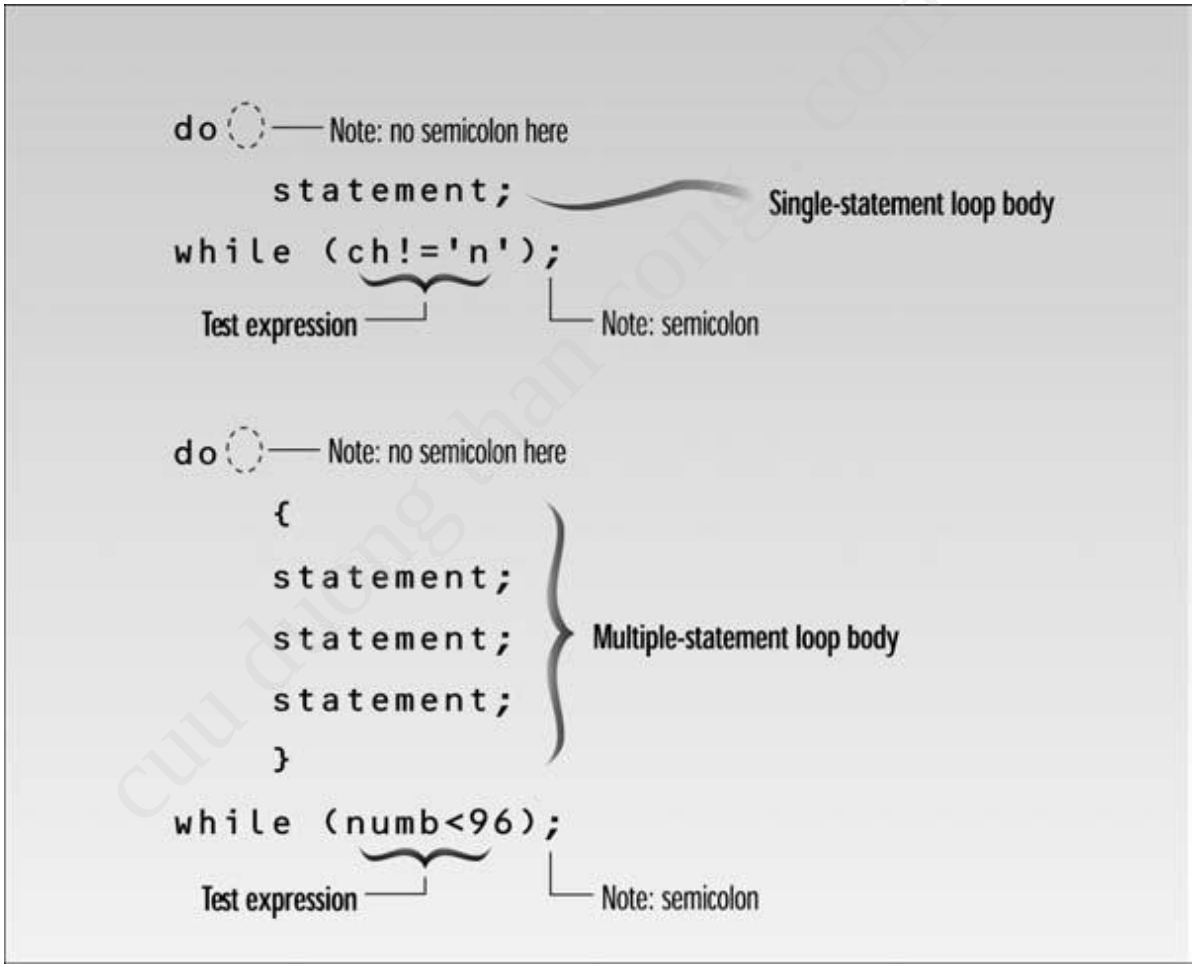
The while Loop

```
const unsigned long limit = 4294967295;
unsigned long next=0;
unsigned long last=1;
while( next < limit / 2 )
{
    cout << last << " ";
    long sum = next + last;
    next = last;
    last = sum;
}
```

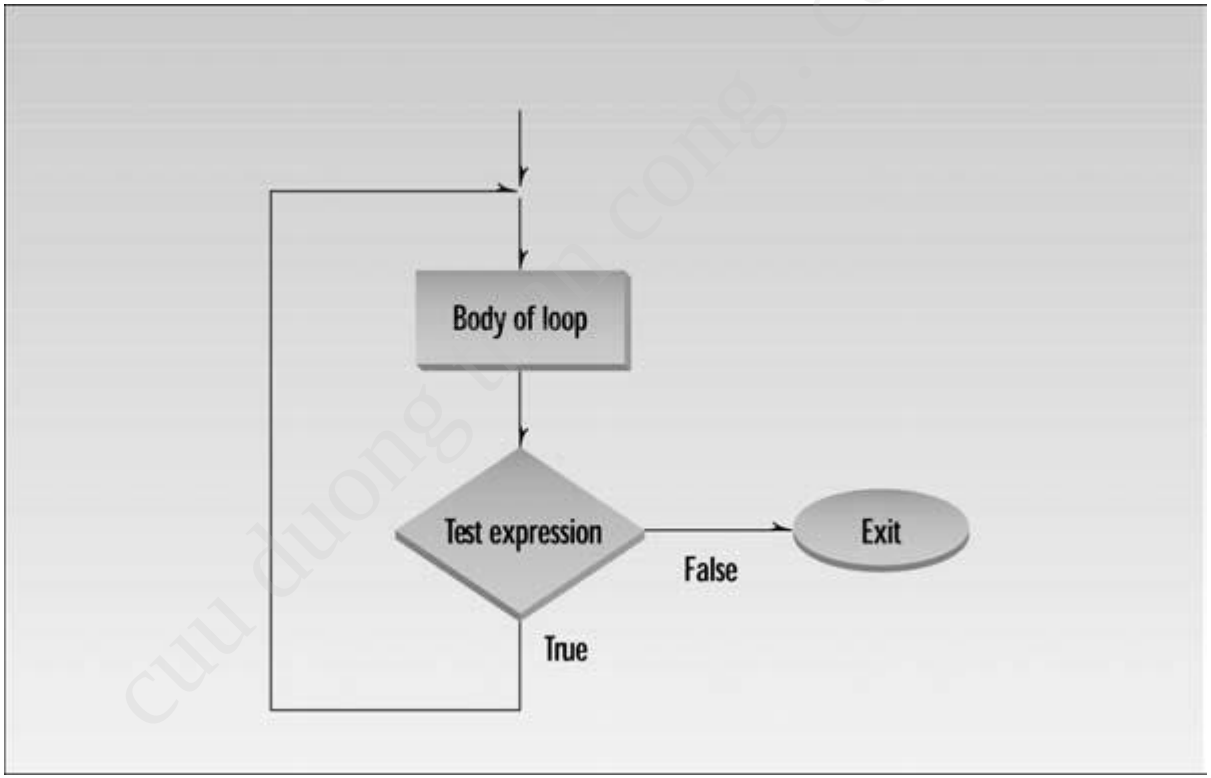
The do Loop

- In a while loop, the test expression is evaluated at the beginning of the loop. If the test expression is false when the loop is entered, the loop body won't be executed at all. In some situations this is what you want. But sometimes you want to guarantee that the loop body is executed at least once, no matter what the initial state of the test expression. When this is the case you should use the do loop, which places the test expression at the end of the loop.

The do Loop



The do Loop



The do Loop

```
do
{
    cout << “Enter dividend: ”; cin >> dividend;
    cout << “Enter divisor: ”; cin >> divisor;
    cout << “Quotient is ” << dividend / divisor;
    cout << “, remainder is ” << dividend % divisor;
    cout << “\nDo another? (y/n): ”;
    cin >> ch;
}
while( ch != ‘n’ );
```

When to Use Which Loop

- The for loop is appropriate when you know in advance how many times the loop will be executed.
- The while and do loops are used when you don't know in advance when the loop will terminate (the while loop when you may not want to execute the loop body even once, and the do loop when you're sure you want to execute the loop body at least once).

Loop Control

- Break
- Continue

Decisions

- Programs also need to make these one-time decisions. In a program a decision causes a onetime jump to a different part of the program, depending on the value of an expression.

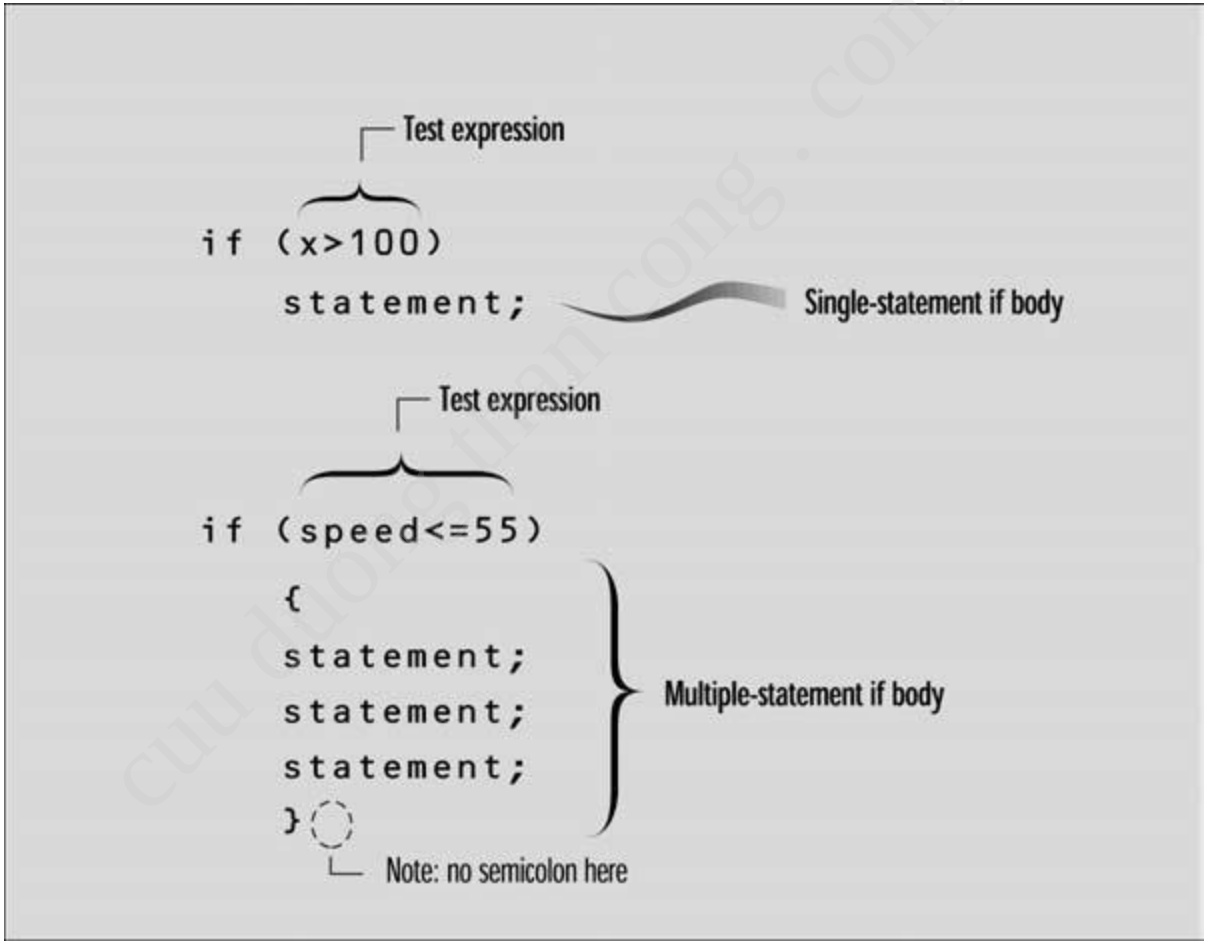
Decisions

- Decisions can be made in C++ in several ways. The most important is with the if...else statement, which chooses between two alternatives.
- Another decision statement, switch, creates branches for multiple alternative sections of code, depending on the value of a single variable.
- Finally, the conditional operator is used in specialized situations.

The if Statement

- The if statement is the simplest of the decision statements. The if keyword is followed by a test expression in parentheses. The statements following the if are executed only once if the test expression is true.

The if Statement



The if Statement

```
int main()
{
    int x;
    cout << “Enter a number: “;
    cin >> x;
    if( x > 100 )
        cout << “That number is greater than 100\n”;
    return 0;
}
```

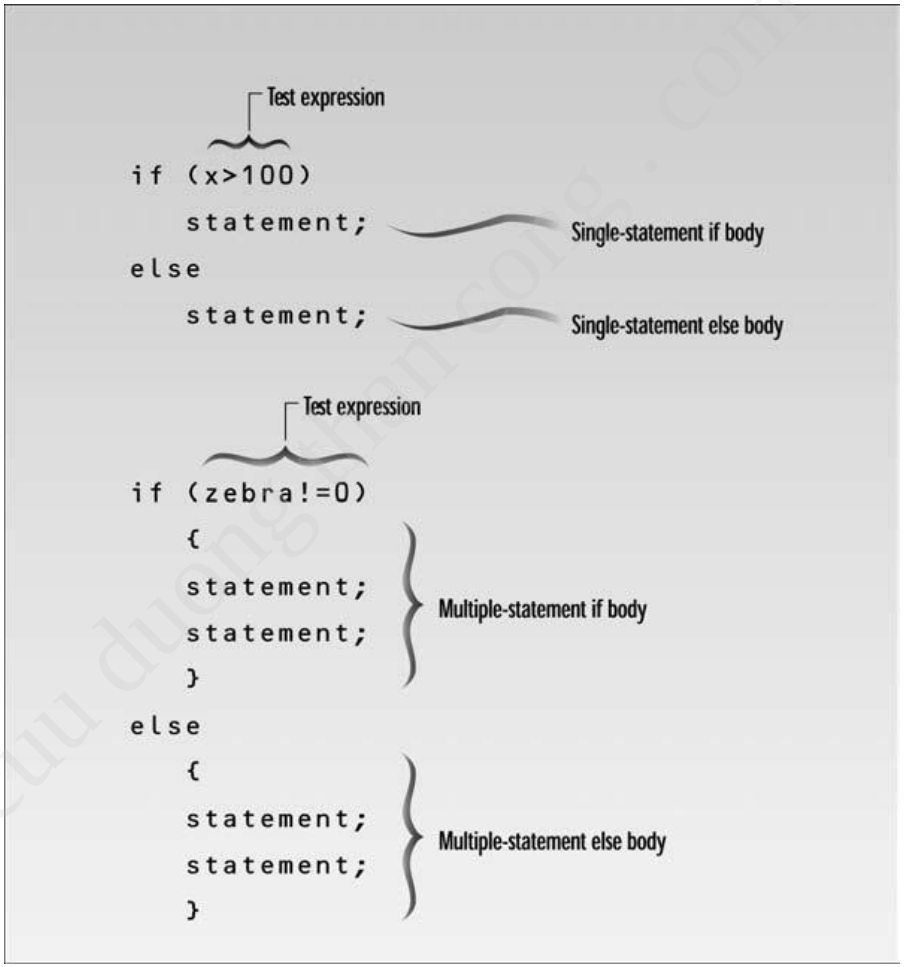
The if Statement

```
for(j=2; j <= n/2; j++)
    if(n%j == 0)
    {
        cout << “It’s not prime”<< endl;
        exit(0);
    }
    cout << “It’s prime”<< endl;
```

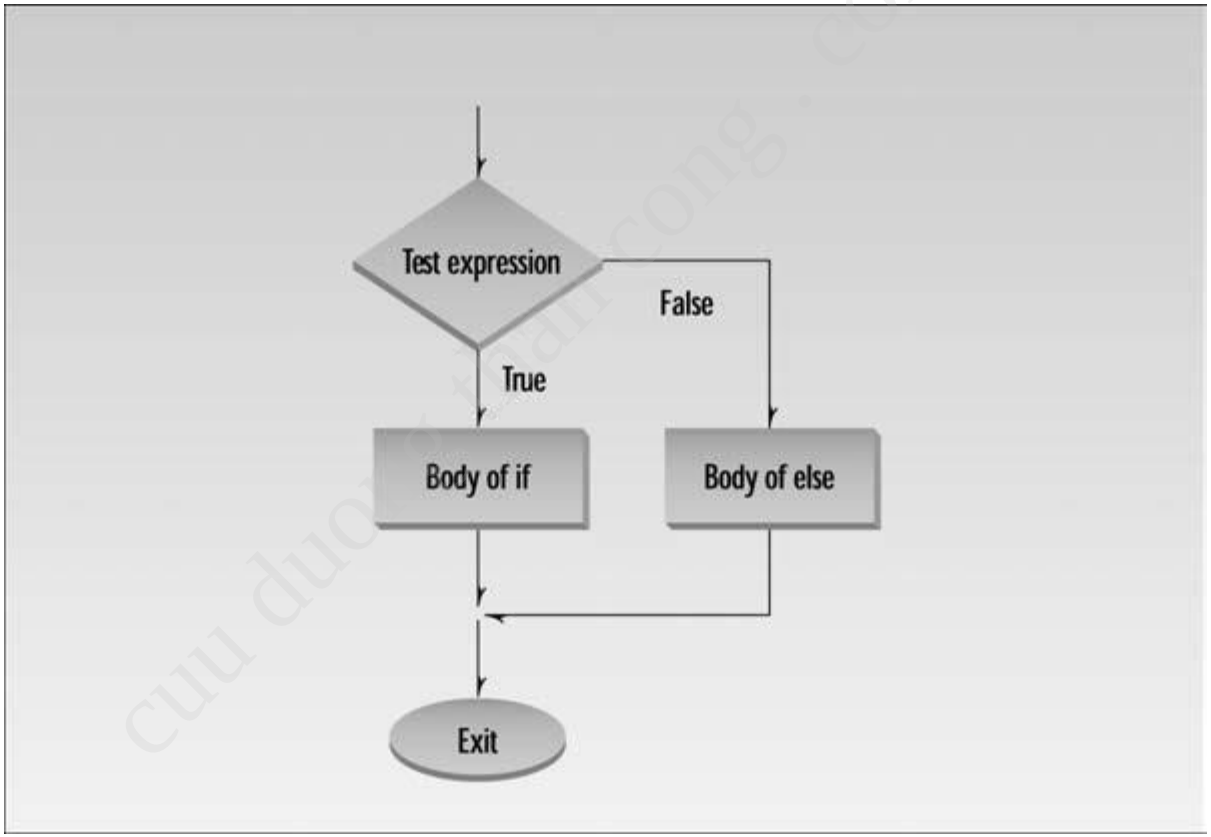
The if...else Statement

- The if statement lets you do something if a condition is true. If it isn't true, nothing happens. But suppose we want to do one thing if a condition is true, and do something else if it's false. That's where the if...else statement comes in.

The if...else Statement



The if...else Statement



The if...else Statement

```
int main()
{
    int x;
    cout << “\nEnter a number: “;
    cin >> x;
    if( x > 100 )
        cout << “That number is greater than 100\n”;
    else
        cout << “That number is not greater than 100\n”;
    return 0;
}
```

The if...else Statement

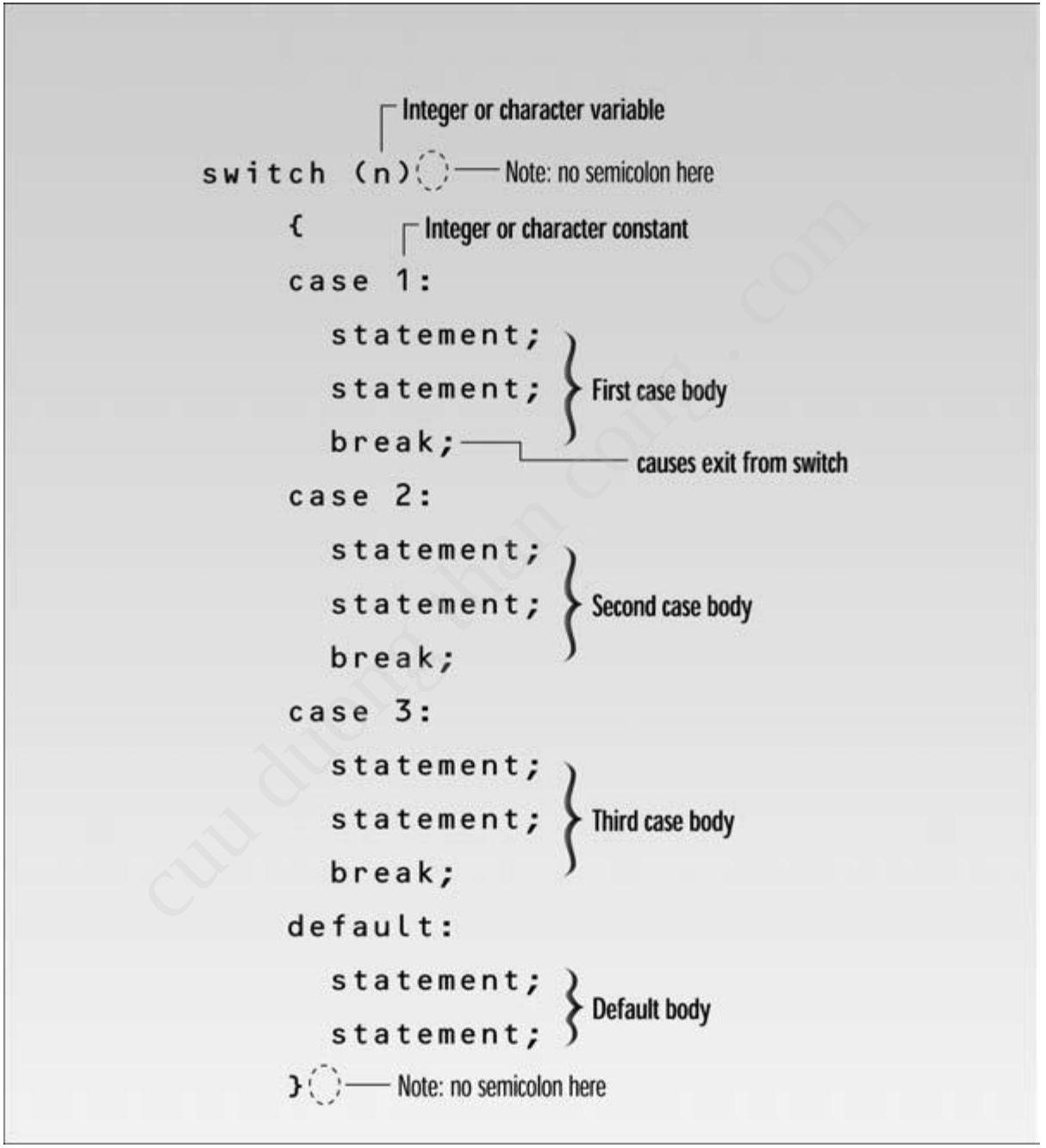
```
while( ch != '\r' )
{
    ch = getche();
    if( ch==' ' )
        wdcount++;
    else
        chcount++;
}
```

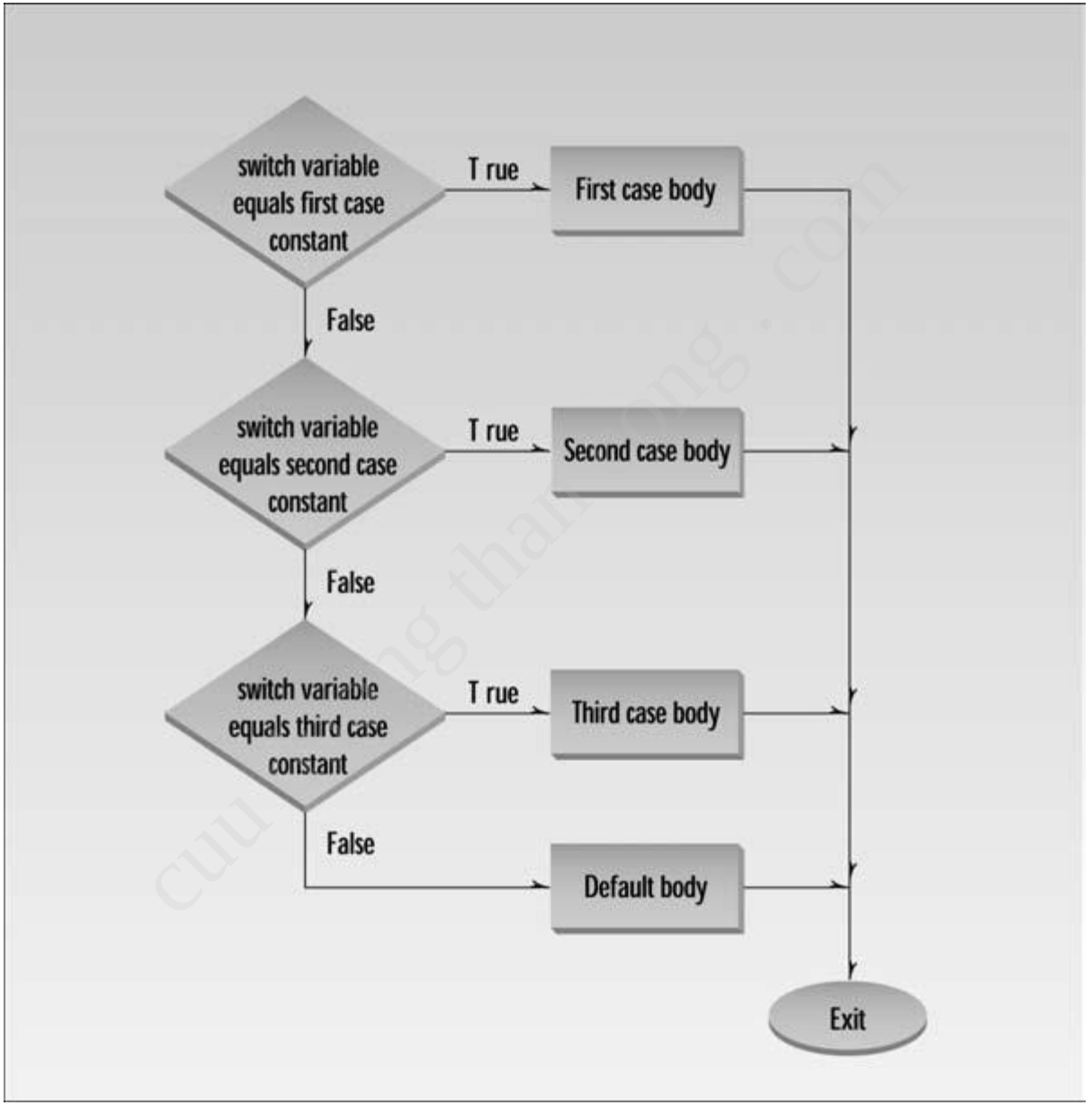
The if...else Statement

```
while( dir != ‘\r’ )
{
    cout << “\nYour location is “ << x << “, “ << y;
    cout << “\nPress direction key (n, s, e, w): “;
    dir = getche();
    if( dir==’n’) y--;
    else if( dir==’s’ ) y++;
    else if( dir==’e’ ) x++;
    else if( dir==’w’ ) x--;
}
```

The switch Statement

- If you have a large decision tree, and all the decisions depend on the value of the same variable, you will probably want to consider a switch statement instead of a ladder of if...else or else if constructions





```
cout << “\nEnter 33, 45, or 78: “;
cin >> speed;
switch(speed)
{
    case 33:
        cout << “LP album\n”;
        break;
    case 45:
        cout << “Single selection\n”;
        break;
    case 78:
        cout << “Obsolete format\n”;
        break;
}
```

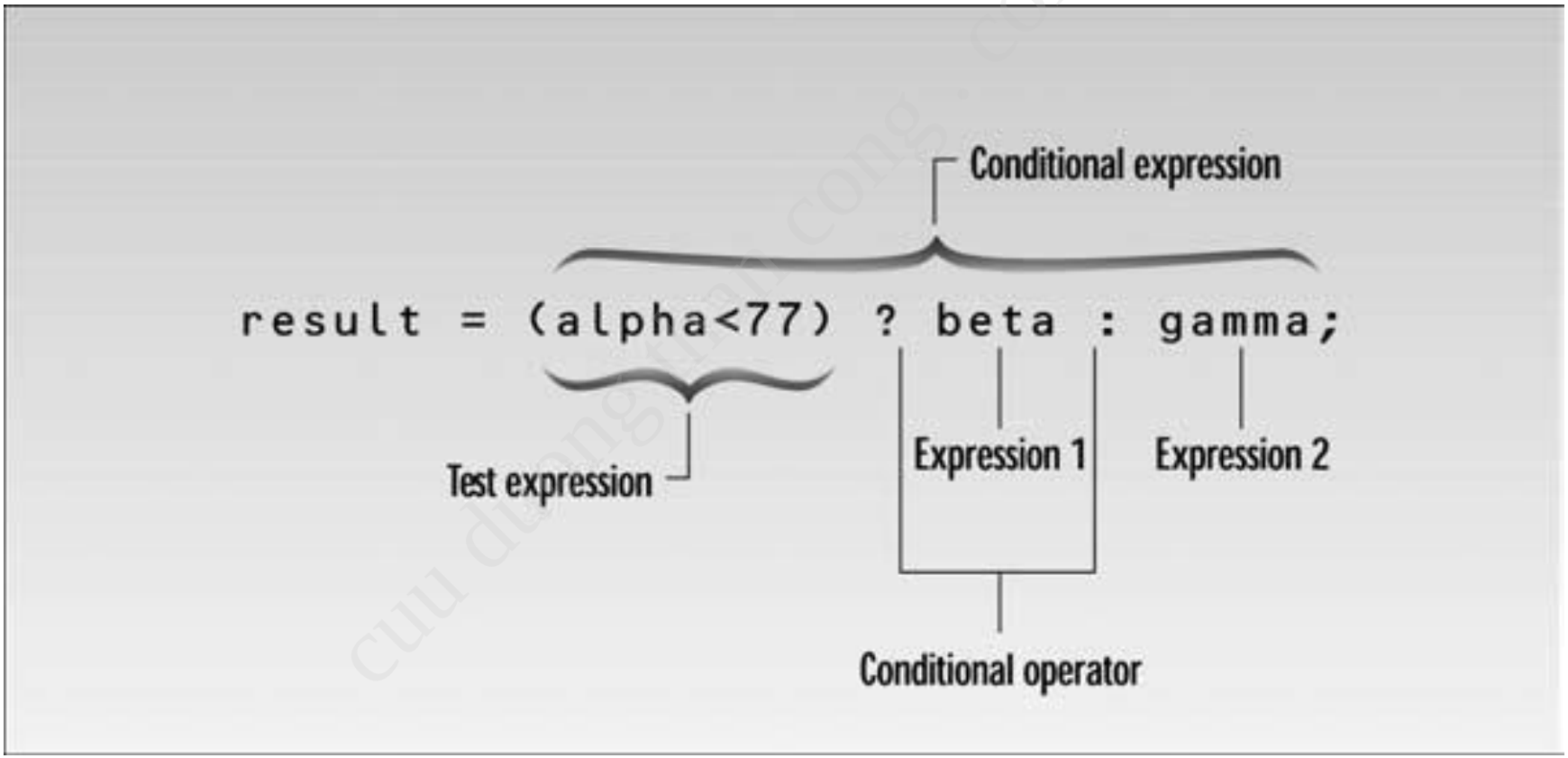
The switch Statement

- The default keywork gives the switch construction a way to take an action if the value of the loop variable doesn’t match any of the case constants.

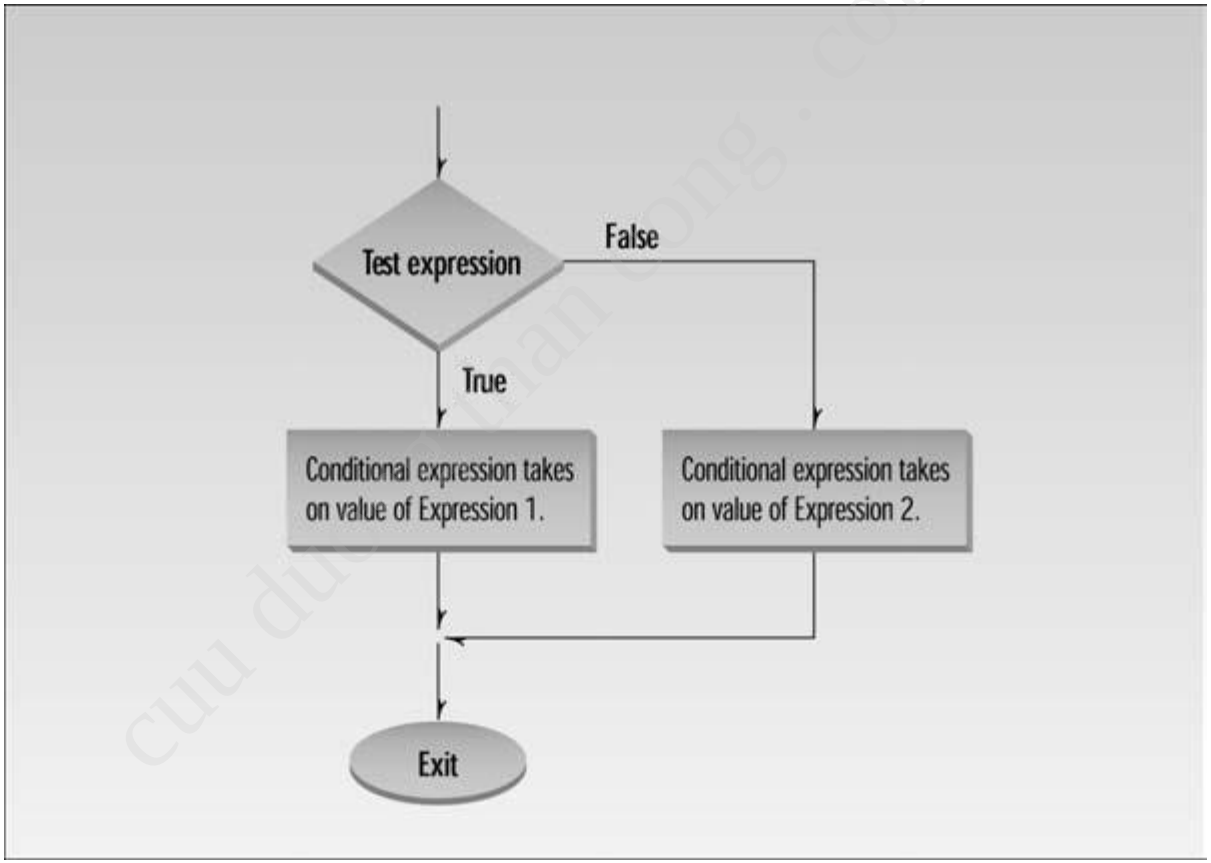
The Conditional Operator

- Here's a strange sort of decision operator. It exists because of a common programming situation: A variable is given one value if something is true and another value if it's false.

The Conditional Operator



The Conditional Operator



Logical Operators

- And (&&)
- Or (||)
- Not (!)

Precedence

Operator type	Operators
Unary	!, ++, --
Arthimetic	*, /, %, +, -
Relational	<, >, <=, >=, ==, !=
Logical	&&, ||
Conditional	?
Assignment	=, +=, -=, *=, /=, %=

The goto Statement

- You insert a label in your code at the desired destination for the goto. The label is always terminated by a colon. The keyword goto, followed by this label name, then takes you to the label.

```
goto SystemCrash;
// other statements
SystemCrash:
// control will begin here following goto
```