南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Advanced Programming

Lab 05

CONTENTS

- ✓ Objectives
- ✓ Knowledge points
- ✓ Exercises

1 CONTENTS

- Learn Relational Expressions
- Master while, do-while and for loops
- Learn logical operators
- Master branching statements
- Master switch multi-branch statement
- Master the use of break and continue statements
- File operation

2 Knowledge Points

2.1 Relational Operators

2.2 Repetition Control Structure

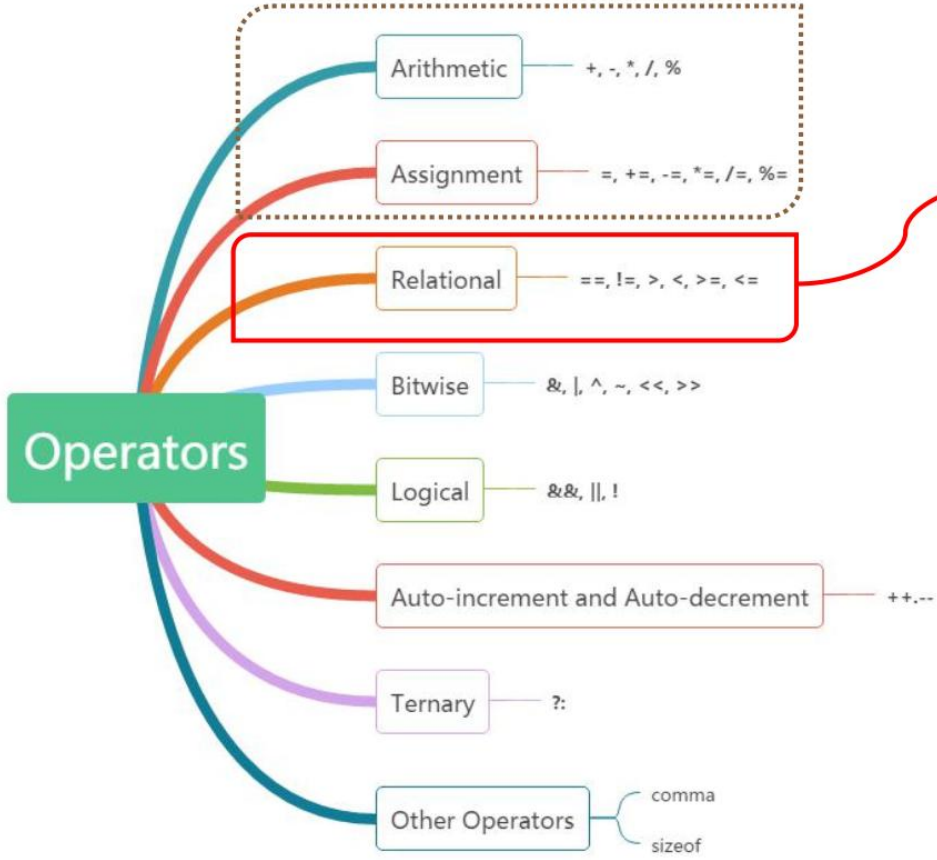
2.3 Logical Operators

2.4 Selection Control Structure

2.5 continue and break statement

2.6 File input/output

2.1 Relational operators



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

The values of relational expressions is **0 for false** or **1 for true** by default. You can set the formatting of the output using **boolalpha** manipulator or setf().
setf(ios_base::boolalpha);

Relational Operator Sample:

```
#include <iostream>
using namespace std;
int main()
{
    int a = 5,b = 2,c = 10;
    cout << " a > b ? " << (a > b) << ",b > c ? " << ( b > c ) << endl;
    cout << "Print the values of relational expressions as boolean formatting:" << endl;
    cout << boolalpha;
    cout << "a > b ? " << (a > b) << ",b > c?" << (b>c) << endl;
    cout << "a * b ==c ? " << (a*b == c) << endl << endl;

    cout << "b - a = "<<( b - a ) << ",its boolean value:" << (bool)( b - a ) << endl;
    cout << "The value of(a = b/c )is: "<< (a = b/c) << " ,its boolean value:" << (bool)(a = b/c) << endl;
    cout << noboolalpha;
    cout << "a == b/c?" <<(a == b/c)<<boolalpha<<",print in logical value of ( a = b/c ):"<<(a == b/c) << endl;
    return 0;
}
```

You can convert the values of arithmetic expressions to bool type explicitly. 0 for false and non-zero for true.

Result:

```
 a > b ? 1,b > c ? 0
Print the values of relational expressions as boolean formatting:
a > b ? true,b > c?false
a * b ==c ? true

b - a = -3,its boolean value:true
The value of(a = b/c )is: 0 ,its boolean value:false
a == b/c ? 1,print in logical value of ( a = b/c ):true
```

2.2 Repetition Control Structure

Difference between while and do-while loop

- **while:** The loop condition is tested at the beginning of the loop before the loop is performed.
- **do-while:** The loop condition is tested after the loop body is performed. Therefore, the loop body will always execute at least once.

```
#include <iostream>
using namespace std;
int main()
{
    int n = 0;
    while (n!=0)
    {
        cout << "this is while loop!"<<endl;
    }
    return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
    int n = 0;

    do{
        cout << "this is do while loop!"<<endl;
    }while (n!=0);
    return 0;
}
```

What's the difference?

Difference between for and while loop

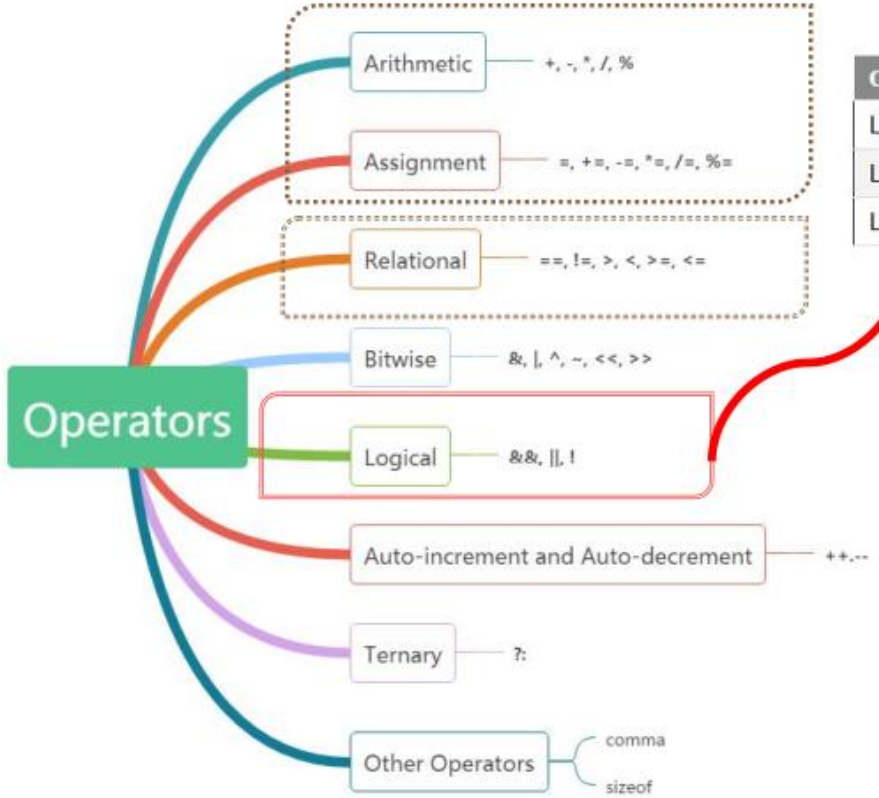
They can both do the same things, but in general if you know how many times you will loop, use a for, otherwise, use a while.

```
for(int i =0; i < 3; i++)
{
    // this goes around 3 times
}
```

```
bool again = true;
char ch;

while(again)
{
    cout << "Do you want to go again(Y/N)?";
    cin >> ch;
    if(ch == 'N')
        again = false;
}
```

2.3 Logical Operator



Operator	Symbol	Form	Operation
Logical NOT	!	!x	true if x is false, or false if x is true
Logical AND	&&	x && y	true if both x and y are true, false otherwise
Logical OR	||	x || y	true if either x or y are true, false otherwise

&& OPERATOR (and)		
a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

|| OPERATOR (or)		
a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

Logical Operators (!, &&, ||)

```
#include <iostream>
using namespace std;
int main()
{
    int a =0,b=3,c =10;
    cout <<"(a && b)=" << (a &&b) << ",(a || b)=" << (a || b) << endl;
    cout << boolalpha;
    cout << "(a && b)=" << (a && b) << ".(a || b)="<<(a || b)<<endl;
    cout << "!(a && b)=" << (!(a && b)) <<", !(a || b)="<<(!(a || b))<<endl;
    cout << "(b || c && a)=" << (b || c && a) << ",(a && (b || c))=" << (a && (b || c)) << endl;
    return 0;
}
```

Result:

```
(a && b)=0,(a || b)=1
(a && b)=false.(a || b)=true
!(a && b)=true, !(a || b)=false
(b || c && a)=true,(a && (b || c))=false
```

&& has higher precedence than ||.

2.4 Selection Control Structure

if and if-else statement

```
if(opt == 1){
    //add
    result = number1+number2;
}
if(opt == 2){
    //sub
    result = number1-number2;
}
if(opt == 3){
    //multiply
    result = number1*number2;
}
if(opt == 4){
    //divide
    result = number1/number2;
}
```

```
if(opt == 1){
    //add
    result = number1+number2;
}else if(opt == 2){
    //sub
    result = number1-number2;
}else if(opt == 3){
    //multiply
    result = number1*number2;
}else if(opt == 4){
    //divide
    result = number1/number2;
}
```

It's logical fine, but it doesn't work very efficiently.

It's more efficient. Because if opt==1, then the addition is performed, but the rest of the operation are definitely not to be look at.

The Dangling else problem

When if statement is nested inside another if statement, the ***else*** clause always matches the most recent unmatched **if** clause in the same block.

```
int i=1,j=2,k=3;
if (i>j)
   if (j>k)
        cout << "Round 1 : j > k!" << endl;
else
    cout << "Round 1 : j < i" << endl;
```

The complier ignores all indentation and matches the else with the preceding **if**.

To force the **else** clause to match the first ***if*** clause, you must add a pair of braces.

```
int i=1,j=2,k=3;
if (i>j){
   if (j>k)
        cout << "Round 2 : j > k!" << endl;
}
else
{
    cout << "Round 2 : j < i" << endl;
}
```

Difference between if and switch

- Check the Expression: An **if-else-if** statement can test boolean-expressions based on ranges of values or conditions, whereas a **switch** statement tests switch-expressions based only on a single int, enumerated value, byte, short, char. The **switch…case** can only judge the condition of equality, and **if** can judge any condition, such as equal, not equal, greater, less, etc.. If your alternatives involve ranges or floating-point tests or comparing two variables, you should use **if else**.

- switch case is faster than if-else: When the number of branches is large (generally larger than 5), **switch-case** is faster than **if-else-if**.

- Clarity in readability: A **switch-case** looks much cleaner than **if-else-if**.

2.5 Difference between continue and break

The main difference is as follows:

- **break** is used for immediate termination of loop
- **continue** terminate current iteration and resumes the control to the next iteration of the loop

continue

```
while (cin.get(ch))
{
    statement1
    if (ch == '\n')
        continue;
    statement2
}
statement3
```

continue skips rest of loop body and starts a new cycle

break

```
while (cin.get(ch))
{
    statement1
    if (ch == '\n')
        break;
    statement2
}
statement3
```

break skips rest of loop and goes to following statement

2.6 Simple File Input and Output

ofstream: Output stream class to operate on files.

ifstream: Input stream class to operate on files.

fstream: Input/output stream class to operate on files.

class	default mode parameter
ofstream	ios::out
ifstream	ios::in
fstream	ios::in | ios::out

```
#include <iostream>
#include <fstream>

int main() {
    using namespace std;

    ofstream outClientFile;
    outClientFile.open("clients.txt", ios::out);

    //Write “hello CS219” to file
    outClientFile << "hello CS219";
    outClientFile.close();

    return 0;
}
```

Create an ofstream object

The ofstream member function open opens a file and attaches it to an existing ofstream object. ios::out is the default value for the second argument.

Result:

clients.txt

```
1    hello CS219
```

ofstream: Checking state flags

is_open(): tests for some subtle problems that the other forms miss, such as attempting to open a file by using an inappropriate file mode.

```
#include <iostream>
#include <fstream>

int main() {
    using namespace std;

    ofstream outClientFile;
    outClientFile.open("clients.txt", ios::out);
    if (outClientFile.is_open())
    {
        cout <<"Open the file for writing a string:\n";
        outClientFile <<"This is an example.\n";
        outClientFile << "Hello CS219!\n";
        outClientFile.close();
    }else{
        cout << "Can not open file!\n";
        return 1;
    }
    return 0;
}
```

The usual tests for successful opening of a file were the following:

```
if(myfile.fail()) ... // failed to open
if(!myfile.good()) ... // failed to open
if (!myfile) ... // failed to open
if(!myfile.is_open())//failed to open
```

Result:

clients.txt

```
This is an example.
Hello CS219!
```

ifstream:Reading from a text file

```
#include <iostream>
#include <fstream>

int main() {
    using namespace std;

    ifstream inClientFile;
    inClientFile.open("clients.txt");

    if (inClientFile.is_open()) {
        string line;
        while (getline(inClientFile, line)) {
            cout << line << endl;
        }
        inClientFile.close();

    } else {
        cerr << "Can not open file!" << endl;
        return 1;
    }

    return 0;
}
```

clients.txt

```
This is an example.
Hello CS219!
```

Result:

```
This is an example.
Hello CS219!
```

```
ifstream inClientFile;
inClientFile.open("client.txt");
```

Result:

```
Can not open file!
```

fstream: Input and output

```
#include <iostream>
#include <fstream>
int main() {
    using namespace std;
    fstream file;
    file.open("data.txt", ios::out);
    if (file.is_open()) {
        string name;
        int age;
        cout << "Enter your name: ";
        getline(cin, name);
        cout << "Enter your age: ";
        cin >> age;
        file << "The name you input is : " << name << endl;
        file << "The age you input is : " << age << endl;
        file.close();
    } else {
        cerr << "Can not open file for reading!" << endl;
        return 1;
    }
    file.open("data.txt", ios::in);

    if (file.is_open()) {
        string line;
        while (getline(file, line)) {
            cout << line << endl;
        }
        file.close();
    } else {
        cerr << "Can not open file for reading!" << endl;
        return 1;
    }
    return 0;
}
```

Result:

data.txt

```
The name you input is : Stephen Prata
The age you input is : 100
```

```
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week05$ ./a.out
Enter your name: Stephen Prata
Enter your age: 100
The name you input is : Stephen Prata
The age you input is : 100
```

Reference:

https://cplusplus.com/reference/fstream/

https://en.cppreference.com/w/cpp/io/basic_fstream

3 Exercises

1. Write a program that uses a loop to read one word at a time until the word **done** is entered. The program should then report the number of words entered(not counting done). A sample run could look like this:

```
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week05$ ./a.out
Enter words (type 'done' to finish):
hello world CS219
print done
Number of words entered: 4
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week05$ ./a.out
Enter words (type 'done' to finish):
done
Number of words entered: 0
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week05$ ./a.out
Enter words (type 'done' to finish):
hello world
cs219 done start
Number of words entered: 3
```

2.1 Write a program using ***if*** statements to count the number of vowels in text read from cin. (Enter or any other termination character is acceptable.)

```
Enter text (press Enter to finish):
hello world
Number of vowels: 3
```

2.2 Based on 2.1 , using ***switch***statements to count the number of vowels in text read from cin.

```
Enter text (press Enter to finish):
hello world
Number of vowels: 3
a: 0
e: 1
i: 0
o: 2
u: 0
```

3. Write a program that asks user to input a string by keyboard, save the string to a file named f1.txt. Convert the lower case letters into the upper case letters and save to another file named f2.txt. Show the contents of f1.txt and f2.txt on the screen respectively.

Sample output:

```
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week05$ ./a.out
Enter a string: hello world
Contents of f1.txt:
hello world
Contents of f2.txt:
HELLO WORLD
```

f1.txt

```
hello world
```

f2.txt

```
HELLO WORLD
```