CS 31 Discussion 1C

Week 3

TA: Tianxiang (Peter) Li

LA: Kyle Wong

Slides

- http://web.cs.ucla.edu/classes/spring19/cs31/
- TA Links
- Discussion 1C
  - Discussion slides
  - Commands for accessing Linux Server (MacOS/Windows)

Outline

- Increment/decrement operators
- Loop
- String
- Exercises
- Q&A: Project2 questions, project1 grading

Comparison Operators

- Two expressions can be compared using comparison operators
- The result of such an operation is either true or false (a **Boolean value**)

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

```
(7 == 5)      // evaluates to false
(5 > 4)       // evaluates to true
(3 != 2)      // evaluates to true
(6 >= 6)      // evaluates to true
(5 < 5)       // evaluates to false
```

Comparison Operators (the equal sign)

- Difference between = and ==
  - In C++ the single equal sign (=) is used as *assignment operator*

```
totalWeight = oneWeight * numberOfBeans;
temperature = 98.6;
count = count + 2;
```

variable expression

Expression can consist of variable, number, operators, or a mix

  - !=, == are *comparison operators*

```
(7 == 5)      // evaluates to false
(3 != 2)      // evaluates to true
```

Comparison Operators

- it's not just numeric constants that can be compared, but just any value, including variables

int a = 2, b = 3, c = 6;

```
(a == 5)      // evaluates to false, since a is not equal to 5
(a*b >= c)    // evaluates to true, since (2*3 >= 6) is true
(b+4 > a*c)   // evaluates to false, since (3+4 > 2*6) is false
((b=2) == a)  //
```

Comparison Operators

- it's not just numeric constants that can be compared, but just any value, including variables

int a = 2, b = 3, c = 6;

```
(a == 5)      // evaluates to false, since a is not equal to 5
(a*b >= c)    // evaluates to true, since (2*3 >= 6) is true
(b+4 > a*c)   // evaluates to false, since (3+4 > 2*6) is false
((b=2) == a)  // evaluates to true
```

Increment/decrement operators

- Both are *executable statements*
- Pre-increment/decrement

  ++n, --n
- Post-increment/decrement

  n++, n--

Post-Increment/decrement

Order of operation:

- The expression n++ first returns the value of the variable n, then it increases its value by 1

```
int n=2;
int valueOfN = n++;
cout << valueOfN << "\n";
cout << n << "\n";
```

Output:

```
2
3
```

- Post-increment:
  - creates a copy of the object
  - increments the value of the object
  - returns the copy from before the increment

Pre-Increment/decrement

Order of operation:

increment the value of a variable before using it in a expression

```
int a = 1;
int b = ++a;
//Here the value of ‘b’ will be 2 because the value of ‘a’ gets modified before using it in the expression.
```

C++ Operator Precedence

From High to Low:

1. ()
2. a++, a--
3. ++a, --a
4. a*b, a/b, a%b
5. a+b, a-b
6. <,>,<=,>=
7. ==, !=

   Relational operators
8. Bitwise operators (AND>XOR>OR)
9. Logical operators (AND(&&) > OR(||))
10. +=, -=, *=, /=, %= → compound assignment

https://en.cppreference.com/w/cpp/language/operator_precedence

Exercise

- x = y = z;
- z *= ++y + 5;
- a || b && c || d;

Exercise

- x = y = z;
- z *= ++y + 5;
- a || b && c || d;

Binary operator ‘=’ has right to left association:

Final answer: x = (y = z);

Exercise

- x = y = z;
- z *= ++y + 5;
- a || b && c || d;

++ has the highest precedence:

+ has the next highest precedence:

Exercise

- x = y = z;
- z *= ++y + 5;
- a || b && c || d;

++ has the highest precedence:

z *= (++y) + 5;

+ has the next highest precedence:

Final answer: z *= ((++y) + 5);

Exercise

- x = y = z;
- z *= ++y + 5;
- a || b && c || d;

Binary operator && has higher precedence than ||

Exercise

- x = y = z;
- z *= ++y + 5;
- a || b && c || d;

Binary operator && has higher precedence than ||:

a || (b && c) || d;

Binary operator || has left to right association:

Final answer: (a || (b && c)) || d;

A common mistake: Write down `if` but really means `else if`

```
int main() {
    int score;
    cout << "Enter a score: ";
    cin >> score;

    if (score >= 90)
        cout << "Grade: A" << endl;
    if (score >= 80)
        cout << "Grade: B" << endl;
    if (score >= 70)
        cout << "Grade: C" << endl;
    else
        cout << "Failed" << endl;

    return 0;
}
```

```
int main() {
    int score;
    cout << "Enter a score: ";
    cin >> score;

    if (score >= 90)
        cout << "Grade: A" << endl;
    else if (score >= 80)
        cout << "Grade: B" << endl;
    else if (score >= 70)
        cout << "Grade: C" << endl;
    else
        cout << "Failed" << endl;

    return 0;
}
```

- Which version of grading programs is correct?

A common mistake: Write down `if` but really means `else if`

```
int main() {
    int score;
    cout << "Enter a score: ";
    cin >> score;

    if (score >= 90)
        cout << "Grade: A" << endl;
    if (score >= 80)
        cout << "Grade: B" << endl;
    if (score >= 70)
        cout << "Grade: C" << endl;
    else
        cout << "Failed" << endl;

    return 0;
}
```

```
int main() {
    int score;
    cout << "Enter a score: ";
    cin >> score;

    if (score >= 90)
        cout << "Grade: A" << endl;
    else if (score >= 80)
        cout << "Grade: B" << endl;
    else if (score >= 70)
        cout << "Grade: C" << endl;
    else
        cout << "Failed" << endl;

    return 0;
}
```

- What if we enter 95?
  - The left one will print 3 lines (Grade A, Grade B, and Grade C), which is not desired

Loops

- Three basic statements: while, do-while, for
- Similar terms to other programming languages
- **Loop Body**
  - Repeated code in a loop
- **Iteration**
  - Each repetition of the loop

Syntax of for loop

for (Initial value ; Execution condition ; Increment value) {

Line 1
Line 2
Line 3
Line 4
Line 5

}

- Execution flow:

Initial | Condition | Block | Increment | Condition | Block | Increment →

→ Condition | Block | Increment | Condition | Block | Increment | Condition

Syntax of while loop

```
while ( Execution condition ) {
        Line 1
        Line 2
        Line 3
        Line 4
        Line 5
}
```

- Execution flow:

Condition | Block | Condition | Block | Condition | Block | Condition

Syntax of do-while loop

```
do {
    Line 1
    Line 2
    Line 3
    Line 4
    Line 5
} while ( Execution condition );
```

- Important reminder: Don't forget the semicolon in the end of while clause!
- Execution flow:

Block | Condition | Block | Condition | Block | Condition

Comparison of 3 types of loop

```
for ( Initial ; Condition ; Increment ) {
    Line 1
    Line 2
}
```

- For
  - When the task is sequential
  - A typical usage is for counting

```
while ( Condition ) {
    Line 1
    Line 2
}
```

- While
  - We don't know how many times we have to repeat the block, but know when to repeat (or stop)

```
do {
    Line 1
    Line 2
} while ( Condition );
```

- Do-while
  - Similar to while, but if we must execute the block at least once
  - Don't forget the semicolon!

Loop replacement

- For -> While

```
for ( Initial ; Condition ; Increment ) {
    Line 1
    Line 2
}
```

```
while (                ) {

}
```

Loop replacement

- For -> While

```
for ( Initial ; Condition ; Increment ) {
    Line 1
    Line 2
}
```

```
while ( Condition ) {
    Line 1
    Line 2
}
```

Loop replacement

- For -> While

```
for ( Initial ; Condition ; Increment ) {
    Line 1
    Line 2
}
```

```
Initial
while ( Condition ) {
    Line 1
    Line 2
    Increment
}
```

Loop replacement

- While -> For

```
while ( Condition ) {
    Line 1
    Line 2
}
```

```
for (        ;        ;        ) {

}
```

Loop replacement

- While -> For

```
while ( Condition ) {
    Line 1
    Line 2
}
```

```
for (        ; Condition ;        ) {
    Line 1
    Line 2
}
```

Nested for loop

```
int main() {
    for (int r = 1; r <= 3; r++) {
        for (int c = 1; c <= 4; c++) {
            cout << "*";
        }
        cout << endl;
    }

    return 0;
}
```

What’s the output?

for loop

```
int main() {
    for (int r = 1; r <= 3; r++) {
        for (int c = 1; c <= 4; c++) {
            cout << "*";
        }
        cout << endl;
    }

    return 0;
}
```

- Execution result:

```
timestring@Bos-MacBook-Pro:/tmp$ ./rect
****
****
****
```

for loop

How do we print out a triangle using for loop?

```
*
* *
* * *
* * * *
* * * * *
```

for loop

```
*
* *
* * *
* * * *
* * * * *
```

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

int main()
{
    int rows;

    cout << "Enter number of rows: ";
    cin >> rows;

    for(int i = 1; i <= rows; ++i)
    {
        for(int j = 1; j <= i; ++j)
        {
            cout << "* ";
        }
        cout << "\n";
    }
    return 0;
}
```

outer loop determined the row number
i =1, i=2,…,i=rows

Inner loop determined the column number for each row
j=1, j=2,…, j=i

for loop

How do we print out a triangle using using numbers?

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Modify this code →

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

int main()
{
     int rows;

     cout << "Enter number of rows: ";
     cin >> rows;

     for(int i = 1; i <= rows; ++i)
     {
          for(int j = 1; j <= i; ++j)
          {
               cout << "* ";
          }
          cout << "\n";
     }
     return 0;
}
```

for loops

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

int main()
{
     int rows;

     cout << "Enter number of rows: ";
     cin >> rows;

     for(int i = 1; i <= rows; ++i)
     {
          for(int j = 1; j <= i; ++j)
          {
               cout << j << " ";
          }
          cout << "\n";
     }
     return 0;
}
```

String class

- String (sequence of text)
  "random text"
- Although C++ lacks a simple **data type** to manipulate strings, there is a **string class** used to process strings

```
#include <string>
```

must include string library

```
…
string fruit;
```

declare one variable of type string

```
fruit = "apple";
```

"class" will be discussed later during the course
For now, we can regard class as pre-defined data type, which holds its own data members and member functions

String

- An example of compound type is the String class. Variables of this type are able to store sequences of characters, such as words or sentences
- in order to declare and use objects, need to include header

```
// my first string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystring;
  mystring = "This is a string";
  cout << mystring;
  return 0;
}
```

```
This is a string
```

String (tips)

- strings can be tested to see if they're empty

```
cout << "What is your name? ";
string name;
getline(cin, name);

if (name == "")
    cout << "You didn't type a name!" << endl;
else
    cout << "Hello, " << name << endl;
```

The empty string "" is not the same as a string with only blanks, like " " or " " . (can also use empty() or size())

For this course: **use** getline to read string, **do not** use cin

getline() – read string from terminal

```
// cin with strings
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystr;
  cout << "What's your name? ";
  getline (cin, mystr);
  cout << "Hello " << mystr << ".\n";
  cout << "What is your favorite team? ";
  getline (cin, mystr);
  cout << "I like " << mystr << " too!\n";
  return 0;
}
```

user input

```
What's your name? Homer Simpson
Hello Homer Simpson.
What is your favorite team? The Isotopes
I like The Isotopes too!
```

cin extraction always considers spaces (whitespaces, tabs, new-line...) as terminating the value

To get an entire line from cin, there exists a function, called **getline**, that takes the stream (cin) as first argument

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

int main()
{
    //initialize variables
    string playerName = "", favoriteFood = "";
    int age = 0;

    //get user input
    cout << "What is your name? " ;
    getline(cin, playerName);

    cout << "What is your age? " ;
    cin >> age;

    cout << "What is your favorite food? " ;
    getline(cin, favoriteFood);

    cout << "Welcome to the game " << playerName << ". "
    << "We are glad to hear that you like to eat "
    << favoriteFood << " and are " << age << " years old." << endl;

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

- cin does *not* consume the new line character after the last digit
- A subsequent getline(cin,s) would consume the new line character stored and set s to the empty string without waiting for the user to type anything.

```
What is your name? Prof Gustin
What is your age? 22
What is your favorite food? Welcome to the game Prof Gustin. We are glad to hear that you
like to eat  and are 22 years old.
```

```
//get user input
cout << "What is your name? " ;
getline(cin, playerName);

cout << "What is your age? " ;
cin >> age;
cin.ignore(100, '\n');

cout << "What is your favorite food? " ;
getline(cin, favoriteFood);

cout << "Welcome to the game " << playerName
<< "We are glad to hear that you like to eat
<< favoriteFood << " and are " << age << " ye

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

100 Read and discard 100 characters,

'\n' Recognize and discard up until it finds that particular character,

Function: ignore 100 characters or until it gets to a new line character

cin.ignore () summary

- **Only** use cin.ignore(…) if you read a **number** and the next thing you will be reading is a **string**
- **Don't** use cin.ignore(…) if you read a **string** and the next thing you'll be reading is a **string**

following up reading the integer immediately calling cin.ignore(...):

```
cin >> k;
cin.ignore(10000, '\n');
```

String

Iterating characters in a string

```
string s = "Hello";
for (int k = 0; k != s.size(); k++)
    cout << s[k] << endl;
```

String

```
cout << "Enter some text: ";
string t;
getline(cin, t);

int numberOfEs = 0;
for (int k = 0; k != t.size(); k++)
{
    if (t[k] == 'E'  ||  t[k] == 'e')
        numberOfEs++;
}
cout << "The number of Es (upper and lower case) is "
<< numberOfEs << endl;
```

Length of string

- length(), size()
  - Returns the length of the string, in terms of bytes
  - Both return the exact value
- prefer s.size() to s.length()
  - all other containers in the library have size() but not length()
  - Array, vector, list …

Return character at specific position in string

```
// string::at
#include <iostream>
#include <string>

int main ()
{
  std::string str ("Test string");
  for (unsigned i=0; i<str.length(); ++i)
  {
    std::cout << str.at(i);
  }
  return 0;
}
```

Exercises