Intro to C++ | 1.5

IOStream

Akis Zachariadis - Games Dept. Coordinator

API

- ✓ **istream** | A class that deals with input streams.
  - ✓ Used with extraction operator (>>)
- ✓ **ostream** | A class that deals with output streams
  - ✓ Used with the insertion operator (<<)
- ✓ **iostream** | A class that handles both input and output
- ✓ **Standard Streams** | cin, cout, cerr, clog

Reading input with istream

- Limit input stream
  - We need a string manipulator. We can use them by importing the iomanip header.
  - The class setw(10) extracts from the input stream the first 10 characters
  - If we type 'helloworldthisisdarkness' the outcome will be: 'helloworld'
- Ignore whitespaces
  - We must use the cin.get() method. It extracts the first char from the input stream
  - If we type 'hello world' the outcome will be: 'helloworld'

```
//limit the input
// include <iomanip>
char buf[10];
std::cin >> std::setw(10) >> buf;

std::cout << buf << std::endl;

//not ignoring whitespaces
char ch;
while(std::cin >> ch){
   std::cout << ch;
}

//ignore whitespaces
while(std::cin.get(ch)){
   std::cout << ch;
}
```

Reading input with istream

- Limit input stream (string version)
  - std::cin.get(char[] , int num)
  - Be careful in our example we don't read 11 char but 10. The last one is used for the terminator
  - Also, cin.get() does not read the "\n", use cin.getline() instead.
- Get the number of characters extracted from input stream
  - Use the cin.gcount().
- Read std::string with getline()
  - Use the special version of std::getline(). It is included on the header <string>

```
//limit input buffer with cin.get(string)
char strBuff[11];
std::cin.get(strBuff, 11);
std::cout << strBuff << std::endl;

//using getline instead of get
std::cin.getline(strBuff, 11);
std::cout << strBuff << std::endl;

//getting the number of char
std::cin.getline(strBuff, 11);
std::cout << strBuff << std::endl;
std::cout << std::cin.gcount() << std::endl;

//string version of getline()
std::string strBuf;
std::getline(std::cin, strBuf);
std::cout << strBuf << std::endl;
```

Some useful istream functions

- Ignore() – discards the first char in the stream
- Ignore(int) – discards the n first chars in the stream
- Peek() – allows to read a char from the stream without removing it
- Unget() – returns the last char read back into the stream to be read
- Putback(char) – get back to the stream a specific char

Formatting

- There are 2 ways to format a string. We can use flags or manipulators
    - Flags are Boolean variables that can be turned on or off.
    - Manipulators are objects placed in a stream and affects only the specific one.
    - In the first example we enable the sign flag and then we disable it
- With manipulators
    - We can import manipulators directly to the output stream
- Some useful formatters
    - Boolalpha – is set prints true or false otherwise prints 0 or 1
    - Showpos – Shows the sign in positive numbers
    - Uppercase – Use uppercase letters
    - Hex,dec,octal – Prints values in hexademical, decimal or octal

```
std::cout.setf(std::ios::showpos);
std::cout << 5 << std::endl;
std::cout.unsetf(std::ios::showpos);
std::cout << 7 << std::endl;
//this will print +5, 7

// Turn on std::ios::hex as the only std::ios::basefield flag
std::cout.setf(std::ios::hex, std::ios::basefield);
std::cout << 5 << '\n';

std::cout << std::hex << 5 << std::endl;
std::cout << 8 << std::endl;
std::cout << std::dec << 9 << std::endl;
//this will print hex, hex, dec
```

More formatters

- Precision
  - Std::fixed – Use decimal notation
  - Std::scientific – Use scientific notation
  - Std::showpoint – Show a decimal point and trailing for 0
  - Std::setprecision(int) – sets the precision of floating numbers
  - Std::precision() – returns the current precision

```
std::cout << std::fixed << '\n';
std::cout << std::setprecision(3) << 111.589 << '\n';
std::cout << std::setprecision(4) << 111.589 << '\n';
std::cout << std::setprecision(5) << 111.589 << '\n';
std::cout << std::setprecision(6) << 111.589 << '\n';
std::cout << std::setprecision(7) << 111.589 << '\n';
//this outputs: 111.589, 111.5890 etc.

std::cout << std::setprecision(3) << 111.589 << '\n';
std::cout << std::setprecision(4) << 111.589 << '\n';
std::cout << std::setprecision(5) << 111.589 << '\n';
std::cout << std::setprecision(6) << 111.589 << '\n';
std::cout << std::setprecision(7) << 111.589 << '\n';
//this outputs: 111, 111.5 etc.
```

String width and Justification

- Manipulators – internal, left, right, setfill(char), setw(int)
- Functions – fill(), fill(char), width(), width(int)
- If we want to fill the space use fill(char)

```cpp
std::cout << -98765 << '\n'; // print default value with no field width
std::cout << std::setw(10) << -98765 << '\n'; // print default with field width
std::cout << std::setw(10) << std::left << -98765 << '\n'; // print left justified
std::cout << std::setw(10) << std::right << -98765 << '\n'; // print right justified
std::cout << std::setw(10) << std::internal << -98765 << '\n'; // print internally justified
//this print:
//-98765
//    -98765
//-98765
//    -98765
//-    98765

std::cout.fill('*');
std::cout << std::setw(10) << -98765 << '\n'; // print default with field width
//****-98765
```

Stream classes

- In order to use a stream class you need to include sstream header
- Use the extraction and insertion operators to work with string streams.

```
std::stringstream os;
//insert some string element intro the stringstream
os << "I'm hungry!" << '\n';
//or
os.str("I'm hungry!");

//And get the data
std::cout << os.str();

//or
std::string strValue;
os >> strValue;
std::cout << strValue;
```

Stream classes

- Convert numbers to string
  - Use the extraction operator (>>)
- Erase the stringstream buffer
  - Use the os.str('') function or string{}

```
//conversion from numerical to string
int value = 50;
double value2 = 50.50;

std::stringstream os;
os << value << ' ' << value2;

std::string strValue, strValue1;

os >> strValue >> strValue1;
std::cout << strValue << ' ' << strValue1;
//outputs 50 50.50

//clear the buffer
os.str("");
//or
os.str(std::string{})
```

```
//conversion from string to numerical
os << "1234567 50.50";
int nvalue;
double nvalue2;

os >> nvalue >> nvalue2;

std::cout << nvalue << ' ' << nvalue2;
//prints 123456 50.50
```

Input Validation

- Numerical Validation
  - We will use the cin.fail() to check if user inputs something different than a numerical
  - We must use the cin.ignore in order to ignore the char from the Enter of the user
  - If the cin.fail is true then we go to the conditional. Then we must clear the buffer and ignore the enter
  - If the gcount is 1 then the input from the user is valid. If gcount is 2 or more then something weird the user typed so we asked him again for his age.

```
while(true){
  std::cout << "Enter your age";
  std::cin >> newAge;

  if(std::cin.fail()) {
    std::cin.clear();
    std::cin.ignore(32767, '\n');
    continue;
  }
  std::cin.ignore(32767, '\n');
  if(std::cin.gcount()>1){
    continue;
  }
  if(newAge <= 0){
    continue;
  }
  break;
}
```

File output

- In order to save values to a file we must include the fstream header.
- Then using the class ofstream we can save a file to our project directory.

```
std::ofstream  outf {"Test.txt"}; //Creates a file with the name test.txt

if(!outf){
   std::cerr << "Oh, you have a mac, you can't read txt files" << std::endl;
   return 1;
}

outf << "I need to eat" << '\n';
outf << "I need to eat meat" << '\n';
```

File input

- In order to import a file from the project directory we need to use the ifstream class
- We can use the stringstream or the std::string in order to save the
- When ifstream goes outofscope destructor will close the file

```
std::ifstream inf{"Test.txt"};

if(!inf){
    std::cerr << "Oh, you have a mac, you can't read txt files" << std::endl;
    return 1;
}

while (inf){
    std::string strInput;
    inf >> strInput;
    std::cout << strInput << '\n';
}

//We can close explicit the file
inf.close();
return 0;

//When inf goes out of scope, the ifstream will call the
// destructor and will close the file
```

Challenge #6

Να δημιουργηθεί πρόγραμμα που θα διαβάζει τα αποτελέσματα από τους γύρους στο Battle Simulator και θα τα αποθηκεύει σε ένα αρχείο Score.dat. Επίσης θα δίνεται η δυνατότητα να εκτυπώνει στην κονσόλα τα αποτελέσματα του προηγούμενου battle.