OntarioTech Science

Kourosh Davoudi
kourosh@uoit.ca

Week 2: Input/output in C++

CSCI 1061: Programming Workshop II

Learning Outcomes

In this week, learn:

- Command line arguments
- Format output in C++
- File I/O

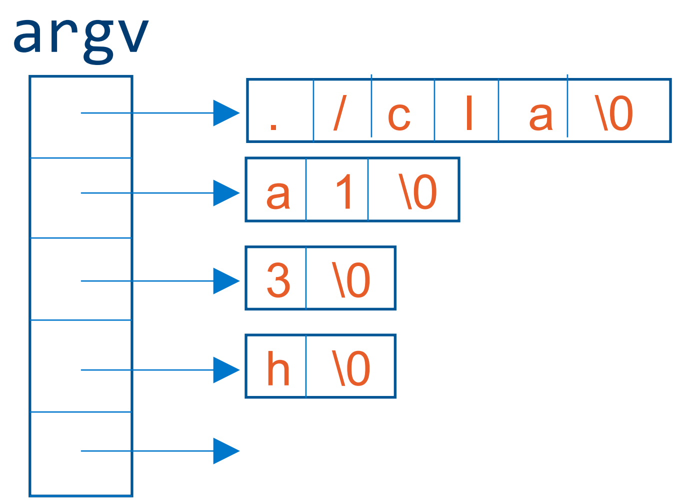
Command Line Argument

- What is command line arguments
  - ./prog_name — Executing the program without argument
  - /prog_name $\underbrace{\text{a1 a2 a3}}_{\text{arguments}}$ — Executing the program with some arguments
- Example
  - ./add_nums 2 3

Command Line Argument

```cpp

#include <iostream>

using namespace std;

int main(int argc, char const *argv[])
{
    for (int i = 0; i < argc; i++)
        cout << argv[i] << endl;
    return 0;
}
```

```
./cla a1 3 h hello
```



argc = 5

Command Line Argument

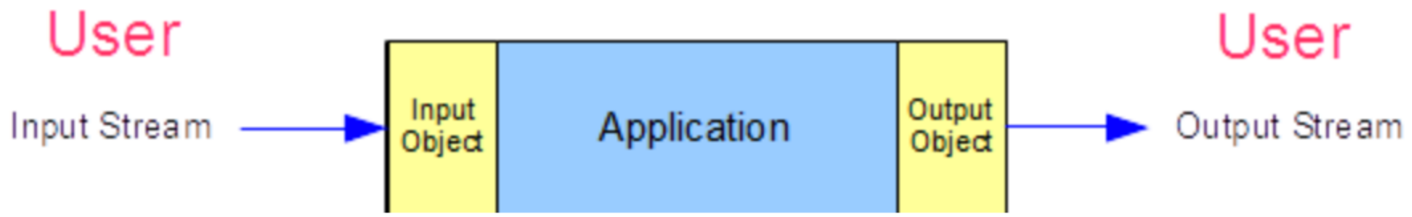
```cpp

#include <iostream>

using namespace std;

int main(int argc, char const *argv[])
{
    for (int i = 0; i < argc; i++)
        cout << argv[i] << endl;
    return 0;
}
```

```
./cla a1 3 h hello
```

Output:

```
./cla
a1
3
h
hello
```

Streams



- The cin object is an instance of the istream type:
  - extracts a sequence of characters from the standard input stream
  - converts that sequence into a specified type and stores that type in system memory.
- The cout object is an instance of the ostream type:
  - copies data from system memory into an output stream;
  - converts the data in system memory into a sequence of characters

istream

- ignore(n =1, delim=EOF)
  - Extracts characters from the input buffer and discards them, until either *n* characters have been extracted, or one compares equal to *delim*.
  - Does not skip leading whitespace.
  - The no-argument version discards a single character

istream methods

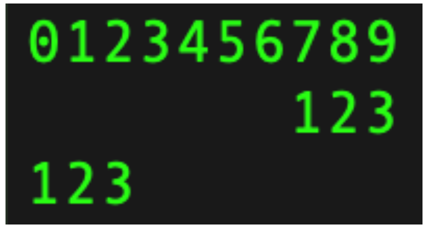
- ignore(n =1, delim=EOF)
  - Extracts characters from the input buffer and discards them, until either *n* characters have been extracted, or one compares equal to *delim*.
  - Does not skip leading whitespace.
  - The no-argument version discards a single character

```
cin.ignore(100,'l');

cin >> str;                                   // "hello"

cout << "|" <<str << "|" << endl;          // "|lo|"  ignore 'hel'
```

ostream methods

- width(int)
  - Specifies the minimum width of the next output field:

```
#include <iostream>

using namespace std;

int main(int argc, char const *argv[])
{

    cout << "0123456789" << endl;

    cout.width(10);

    cout << 123 << endl;

    cout << 123 << endl;

    return 0;
}
```

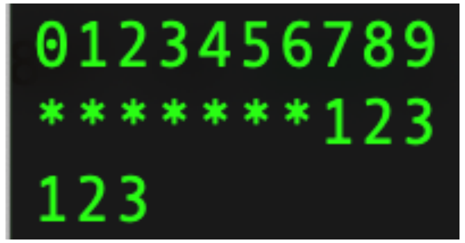


```
0123456789
       123
123
```

ostream methods

- fill(char)
  - Defines the padding character.

```
#include <iostream>

using namespace std;

int main(int argc, char const *argv[])
{

    cout << "0123456789" << endl;

    cout.fill('*');

    cout.width(10);

    cout << 123 << endl;

    cout << 123 << endl;

    return 0;
}
```

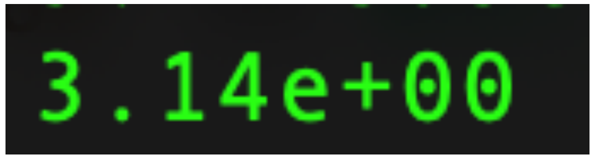


```
0123456789
*******123
123
```

ostream methods

- precision()
  - Sets the precision of subsequent floating-point fields. The default precision is 6 units.

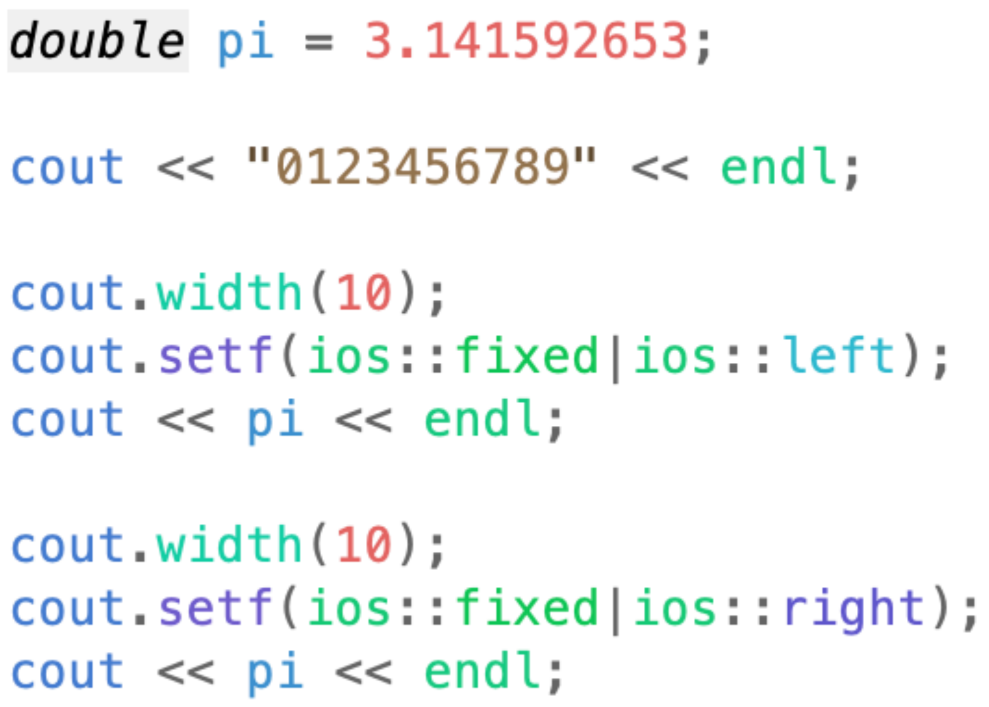
```
cout.precision(2);
cout << pi << endl;
```



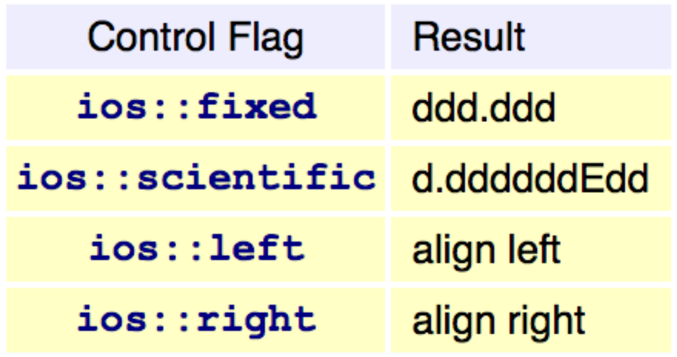
```
3.14e+00
```

ostream methods

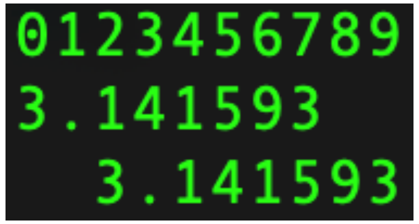
- setf()/unsetf()
  - Control formatting and alignment. Their control flags include:



```cpp
double pi = 3.141592653;

cout << "0123456789" << endl;

cout.width(10);
cout.setf(ios::fixed|ios::left);
cout << pi << endl;

cout.width(10);
cout.setf(ios::fixed|ios::right);
cout << pi << endl;
```



Control Flag	Result
ios::fixed	ddd.ddd
ios::scientific	d.ddddddEdd
ios::left	align left
ios::right	align right



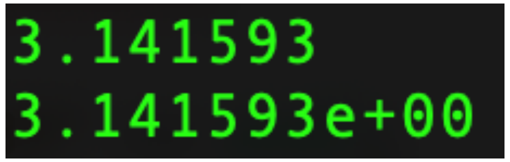
```
0123456789
3.141593
  3.141593
```

ostream methods

- setf()/unsetf()
  - control formatting and alignment. Their control flags include:

Control Flag	Result
`ios::fixed`	ddd.ddd
`ios::scientific`	d.ddddddEdd
`ios::left`	align left
`ios::right`	align right

```
cout.width(10);
cout.setf(ios::fixed|ios::left);
cout << pi << endl;

cout.unsetf(ios::fixed);

cout.width(10);
cout.setf(ios::scientific);
cout << pi << endl;
```



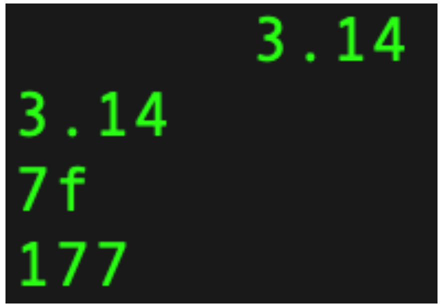
```
3.141593
3.141593e+00
```

Manipulators

- Manipulator defined: "A function called in nontraditional way"
  - Can have arguments
  - Placed after insertion operator
  - Do same things as member functions!
    - In different way

```
#include <iomanip>
```

```
double pi = 3.141592653;

cout << setw(10) << setprecision(3) << pi << endl; // using manipulators
cout << pi << endl;

int x = 127;

cout << hex << x <<  endl;

cout << oct << x <<  endl;
```



```
      3.14
3.14
7f
177
```

Streams Usage

- We've used streams already
  - cin
    - Input stream object connected to keyboard
  - cout
    - Output stream object connected to screen
- Can define other streams
  - To or from files
  - Used similarly as cin, cout

File Connection

- Must first connect *file* to *stream object*
- For input:
  - File → ifstream object
- For output:
  - File → ofstream object
- Classes ifstream and ofstream
  - Defined in library <fstream>
  - Named in std namespace

File I/O Libraries

- To allow both file input and output in your program:

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

OR

```
#include <fstream>
using std::ifstream;
using std::ofstream;
```

File I/O

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

using namespace std;

int main()
{
    ifstream fin;
    ofstream fout;

    fin.open("infile.txt");

    int x,i=0;

    if (fin.fail()) // check if it is successful
    {
        cerr << " Cannot open the input file!" << endl;
        return 1;
    }


    fout.open("outfile.txt");
    if (fout.fail())
    {
        cerr << " Cannot open the output file!" << endl;
        return 1;
    }

    while(fin>>x)  // reading form file
    {
        fout << "Number " << i++ << " is: " << x << endl;
    }

    fin.close();

    fout.close();

     return 0;

 }
```

Open the file

Is it successful?

Reading from a file/
Writing into a file

File Names

- Programs and files
- Files have two names to our programs
  - External file name
    - Also called "physical file name"
    - Like "infile.txt"
    - Sometimes considered "real file name"
    - Used only once in program (to open)
  - Stream name
    - Also called "logical file name"
    - Program uses this name for all file activity

```
ifstream fin;
fin.open("infile.txt");
```

Closing Files

- Files should be closed
  - When program completed getting input or sending output
  - Disconnects stream from file
  - In action:

    ```
    inStream.close();
    outStream.close();
    ```

    - Note no arguments
- Files automatically close when program ends

Appending to a File

- Standard open operation begins with empty file
  - Even if file exists → contents lost
- Open for append:

```
ofstream outStream;
outStream.open("important.txt", ios::app);
```

  - If file doesn't exist → creates it
  - If file exists → appends to end
  - 2nd argument is class *ios* defined constant
    - In <iostream> library, std namespace

Alternative Syntax for File Opens

- Can specify filename at declaration
  - Passed as argument to constructor

```
ifstream fin;
fin.open("infile.txt");
```

=

```
ifstream fin("infile.txt");
```

Checking File Open Success

- File opens could fail
  - If input file doesn't exist
  - No write permissions to output file
  - Unexpected results
- Member function fail()
  - Place call to fail() to check stream operation success

```
fin.open("stuff.txt");
if (fin.fail())
{
    cout << "File open failed.\n";
    exit(1);
}
```

End of File Check with Read

- Read operation (inStream >> next)
- Returns bool value Expression returns true if read successful
  - Returns false if attempt to read beyond end of file
- In action:

```
double next, sum = 0;
while (inStream >> next)
      sum = sum + next;
cout << "the sum is " << sum << endl;
```

Binary files versus Text files

- What is difference?
  - Text files: Operating systems make some change on byte streams when write them onto the file

`\n` $\xrightarrow{\text{Windows}}$ `\r\n`

```
ofstream fout("infile.txt");
```

  - Binary file: No translation happens:

```
ofstream fout("infile.txt”,ios::binary);
```

Random Access Tools

- Opens same as istream or ostream
  - Adds second argument
  - fstream rwStream;
    rwStream.open("stuff", ios::in | ios:: out);
    - Opens with read and write capability
- Move about in file
  - rwStream.seekp(1000);
    - Positions put-pointer at 1000th byte
  - rwStream.seekg(1000);
    - Positions get-pointer at 1000th byte

Random Access Sizes

- To move about → must know sizes
  - sizeof() operator determines number of bytes required for an object:
    - sizeof(s)   //Where s is string s = "Hello”
    - sizeof(10)
    - sizeof(double)
    - sizeof(myObject)

  - Position put-pointer at 100th record of objects:

```
rwStream.seekp(100*sizeof(myObject) – 1);
```