COMP 3479

Object Oriented Programming in C++

Week 1 of 10

Day 2 of 2

Agenda

1. Quiz
2. Operators
3. Constants
4. Console and File IO Part 1

Hello World!

```
#include <iostream>

using namespace std;

int main( )
{
     cout << "Hello world!" << endl;
     return 0;
}
```

Fundamental Types in C++

- Character types like **char**, char16_t, char32_t, wchar_t
- Signed integer types like signed char, short, **int**, long, long long
- Unsigned integer types like (unsigned) char, short, int, long, long long
- Floating-point types like float, **double**, long double
- Boolean (hooray!) called **bool**

Pre-quiz question

- Let's look at the chart from the previous night
- Examine std::numeric_limits<XXX>::min() where XXX is a float
- Is that a **negative** number?

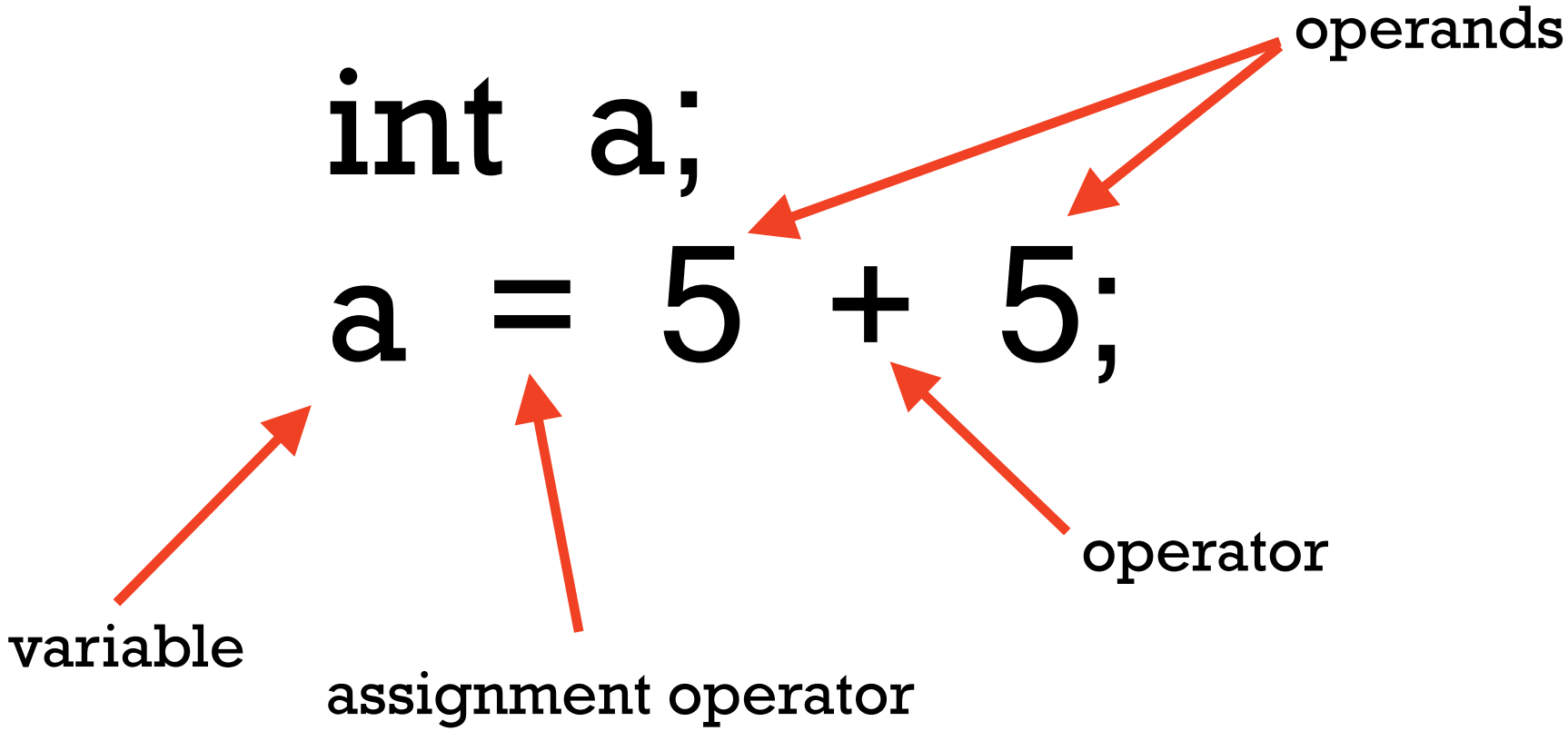
min() v lowest() for floating point numbers

- min() is the tiniest value that can be represented
- lowest() is the least value, i.e., no other value lies to the left of this value on the number line

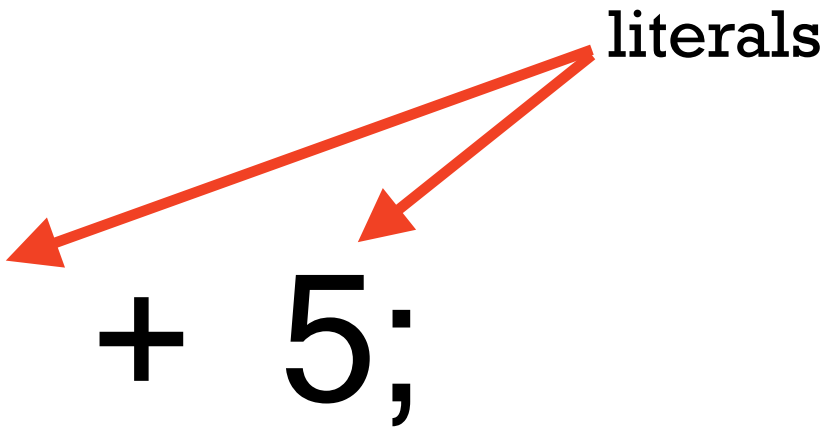
Quiz

1. D2L > Activities > Quizzes
2. Quiz 1
3. 10 minutes
4. Save your work after each question
5. GOOD LUCK!

Operators and operands



Literals



Arithmetic operators

C++ has the usual litany of arithmetic operators:

1. + Addition
2. - Subtraction
3. * Multiplication
4. / Division
5. % Modulo (takes the sign of the dividend, not the divisor)

Compound operators

These should all be familiar to you:

1. +=
2. -=
3. *=
4. /=
5. %=

Compound operators

These might be new (hint: think bits!):

1. >>=
2. <<=
3. &=
4. ^=

Increment and decrement operators

1. ++
2. --

Remember pre vs post!

Relational and comparison operators

1. ==
2. !=
3. >
4. >=
5. <
6. <=

Logical operators

1. !
2. &&
3. ||

Bitwise operators

1. & AND
2. | (the pipe over the \) OR
3. ^ XOR
4. ~ NOT
5. << shifts bits left
6. >> shifts bits right

Some more assorted operators

1. ? : (ternary operator)
2. , (comma operator, yuck)
3. ( ) (casting operator)
4. sizeof
5. And more (later this term)…

Final word about operators

Make sure you are familiar with the **rules of precedence and associativity**:

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

It's time to do some coding

Some important notes about C++ programs:

1. One main method
2. main method may call other functions, just like C
3. In C++, the source file is a .cpp file
4. In C++, the header file is a .hpp file
5. Put function prototypes in the header file
6. Put function definitions in the source file
7. Use #pragma once instead of #ifndef to ensure header file is only included ONCE

IN CLASS ACTIVITY

1. Write a function called fibonacci that accepts an integer n and returns the nth fibonacci number.
2. Write a function called shift that accepts two integers and a boolean. If the boolean is true, bitwise shift the first integer LEFT the number of bits specified by the second integer. If the boolean is false, bitwise shift the first integer RIGHT.
3. CHALLENGE: Write a function called isPrime that accepts an integer and returns true if it is prime and false if it is not prime.
4. Test your code by invoking the functions from the main method. Are there any restrictions for using your functions?

Constants in C++

Old-style – a preprocessor directive:

#define PI 3.1415926535

(Remember: no semi-colon!)

Immutability with const

```
const int some_value{1};
const int some_other_value; // ERROR!
const float pi{3.14159};
const char top_score{'A'};
const bool larger{some_value < pi};
```

It is mandatory to set a constant's value in its declaration!

Useful as a modifier for function parameters.

"I promise not to change this value."

Immutability with constexpr (more later!)

```
constexpr double another_value{1.3};
```

“To be evaluated at compile time.”

Useful for performance:

- Permits placement of data in memory where it is unlikely to be corrupted
- Values known during compilation are privileged and can be placed in read-only memory
- Note that const expressions don’t offer the same guarantee

When to use const vs constexpr

- Will be important when we talk about:
  - Static variables
  - Constructors
- For now:
  - A constexpr must be assigned a value when it is declared
  - A const can be assigned a value after compilation

Example

```
#include <iostream>
using namespace std;
int main()
{
     int input;
     cin >> input; // WOW WHAT'S THIS?
     const int constant_input = input;
     cout << constant_input << endl;
     return 0;
}
```

5 MINUTE BREAK

1. Go stretch
2. Check your phone
3. Prepare for our final hour!

Formatting output: member functions

- Recall std::cout is a **global object** of class **ostream**
- Recall in Java behaviours are called methods
- In C++ we call them **member functions**
- Check out the member functions here: http://en.cppreference.com/w/cpp/io/basic_ostream
- Note the outline format (compare to JavaDoc):
  - Global objects
  - Member types
  - Member functions, member types, non-member functions
  - Inherited types, constants, functions, etc.

Member functions

What do these lines of code do?

```
cout.setf(ios_base::fmtflags);
cout.unsetf(ios_base::fmtflags);
```

The std::ios_base superclass of std::basic_ostream defines ios_base::fmtflags that we can use to format output:

http://en.cppreference.com/w/cpp/io/basic_ostream
http://en.cppreference.com/w/cpp/io/ios_base/fmtflags

Some rules

setf(flag) and unsetf(flag)

- Argument can be:
  - boolalpha
  - showbase
  - uppercase
  - showpos

setf(flag, flag)

- Arguments can be:
  - dec/oct/hex, basefield
  - fixed/scientific, floatfield
  - left/right/internal, adjustfield

Less verbose: output manipulators

- Printing in hex in C with printf requires a lot of typing
- Printing in hex in C++ is almost too easy:

```
int n{15};
cout << hex << n << endl; // f
```

We call these **output manipulators**.

Under the hood

```
ostream& hex(ostream& outputstream)
{
    outputstream.setf(ios_base::hex,
                      ios_base::basefield);
    return outputstream;
}
```

Output manipulators

- **showpos/noshowpos** - assuming *n* is now 123

```
cout << showpos << n;      // +123
cout << noshowpos << n;    // 123
```

- **dec/hex/oct**

```
cout << dec << n; // 123
cout << hex << n; // 7b
cout << oct << n; // 173
```

Output manipulators

- **uppercase/nouppercase**

```
cout << uppercase << hex << n;  // 7B
cout << nouppercase << hex << n; // 7b
```

- **showbase/noshowbase**

```
cout << showbase << hex << n<< endl; // 0x7b
cout << noshowbase << hex << n << endl; // 7b
```

Output manipulators

- **left/internal/right** - assuming *n* is -123

```
cout << setw(6) << left << n;     // |-123   |
cout << setw(6) << internal<< n; // |-   123|
cout << setw(6) << right<< n;     // |   -123|
```

- **showpoint/noshowpoint** - assuming d1 = 100.0 and d2 == 100.12

```
cout << noshowpoint << d1<< " " << d2;
// 100 100.12
cout << showpoint << d1<< " " << d2;
// 100.000 100.120
```

Output manipulators

- **fixed/scientific** - assuming *number* is 123.456789

```
cout << fixed << number;        // 123.456789
cout << scientific << number; // 1.234568E+02
```

- **boolalpha/noboolalpha** - assuming *fun* is true

```
cout << boolalpha << fun;       // true
cout << noboolalpha<< fun;    // 1
```

Output manipulators with <iomanip>

- **setw(value)** sets minimum width for one field only

```
cout << setw(5) << number; // |  123| if n = 123
```

- **setfill(fillchar)**

```
cout << setfill('*') << setw(5) << number;
// prints **123
char c = cout.fill();
cout.fill(c);
```

Output manipulators with <iomanip>

- **setprecision(value)**

```
// assuming number is 123.4567845678
cout << setprecision(7) << number; // 123.4568
streamsize prec = cout.precision();
```

Note: default precision = 6

Member functions vs output manipulators

Member Function	Output Manipulator

`cout.setf`
`    (ios_base::showpos);`
`cout << number;`
	`cout << showpos << number;`

`cout.width(5);` `cout << number;`	`cout << setw(5) << number;`

Q: Which looks easier?

What about input?

- Getting input with Java requires a scanner and a non-trivial amount of code
- Getting input with C is dangerous and requires finesse with fgets and sscanf (recall scanf was not our friend)

- C++: use std::cin

```
int m, n;
cin >> m >> n; // Input 12 34, or 12 <enter> 34
```

Read an int

```
int hours;
cin >> hours;
cout << “Today I slept for “ << hours
     << “hours” << endl;
```

Read a floating point number

```
double weight_kg;
cin >> weight_kg;
cout << “I weigh “ << weight_kg
     << “kilos” << endl;
```

It's not infallible, though!

```
constexpr int first_name_length = 5;
char first_name[first_name_length];
cin >> first_name; // NOOOOOO DON'T DO THIS
```

Recall that char[] == char *

cin doesn't know the length of the array

We have a memory allocation issue

But we can fix it!

```
#include <iomanip>

constexpr first_name_length = 5;
char first_name[first_name_length];
cin >> setw(5) >> first_name;
```

IO: input II

- What if input fails?
- ios_base::iostate contains:
  - ios_base::**failbit** (operation failed)
  - ios_base::**badbit**
  - ios_base::**eofbit** (set on EOF)
  - ios_base::**goodbit** (zero – no bits sets)
- cin is true if cin.fail() is false:

```
int n;
if (cin >> n)…
```

IO: input

- You can test these bits with cin's member functions:
  1. **fail()** – true iff badbit or failbit are set
  2. **bad()** – true iff badbit is set
  3. **eof()** – true iff eofbit is set
  4. **good()** – true iff goodbit it set (no bits are set)

Hint: call cin.clear() after an input failure!

IO: Input Examples

cin >> n
Assume * represents the EOF

User Input	n	failbit	eofbit
123 456	123	Not set	Not set
123*	123	Not set	Set
hello	No change	set	Not set
*	No change	set	set

IO: Ignoring input

- Recall cin is an istream
- std::basic_stream has a member function called **ignore**

```
ignore()    // skips/ignores/tosses 1 char
ignore(128) // skips 128 char or until EOF
ignore(128, '\n') // skips 128 until EOF or '\n'
ignore(LLONG_MAX, '\n') // Throws away LLONG_MAX
                        // char
```

IO: Throwing away an entire line

```
#include <limits>

cin.ignore
     (numeric_limits<streamsize>::max(), '\n');

istream& ignoreline(istream& is)
{
     is.clear();
     is.ignore
          (numeric_limits<streamsize>::max(), '\n');
     return inputstream;
}
```

YOUR LAB

Write a program that:

1. adds integers entered by the user (ignoring input that is not an integer) until the EOF is entered
2. prints the sum.