

Lab 1: Expressions and I/O

Variables and Data Types

Computer programs use variables to store information in memory. In C++, a programmer must declare the type of information that will be stored in a variable. C++ does not provide a default value for variables so until a programmer initializes a declared variable, there will be "junk" stored in it (i.e. whatever leftover value happens to be in memory). This makes referencing an uninitialized variable very dangerous. Some of the data types we will use in this class are:

- **int:** whole numbers
- **float:** real numbers that can include decimal points
- **double:** an extra precise version of floats
- **char:** one character from the alphabet
- **bool:** one true or false value
- **string:** an ordered collection of characters making a word or phrase

```
int number_of_students = 100;
float temperature = 88.3;
char dayOfWeek = 'm';
string school_name = "Texas State University";
bool undergraduate = true;
```

Example 1: several variable declared and initialized

Variable names should be meaningful and the naming convention should be consistent. Names can be in "snake_case" where words are separated by underscores or "camelCase" where words are separated by capitalizing on letter. Both are common but it's confusing to alternate styles within the same program. Variable names that are a single letter are tempting because they are easy to type but it makes a program hard to read.

Using the **const** keyword before a declaration creates a named constant rather than a variable. Constants can only be initialized. Attempting to change the value of a constant will cause an error.

Console I/O

The console lets users interact with a program through text. It is the easiest way to pass information into and out of a program. The **<iostream>** library is used to add console I/O to programs. Some tools available from this library are:

- **cout**: "console out", prints text to the screen
- **cin**: "console in", takes user input as text
- **>>**: "stream insertion", sends text to cout
- **<<**: "stream extraction", takes text out of cin
- **endl**: "end line", breaks the text to a new line.

```
string fName, lName;
cout << "Please enter your first name: ";
cin >> fName;
cout << "Please enter your last name: ";
cin >> lName;
cout << "Your full name is " << lName << ", " << fName << endl;
```

Example 2: a code snippet which takes input from a user and provides output

```
Please enter your first name: Gentry
Please enter your last name: Atkinson
Your full name is Atkinson, Gentry

Process returned 0 (0x0)   execution time : 6.632 s
Press ENTER to continue.
█
```

Example 3: sample output from Example 2

File I/O

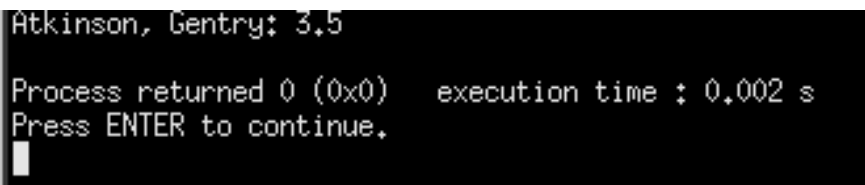
Console I/O is a great way to interact with a user but the output is gone once the program is terminated. Writing output to a file makes it persistent. It can also be tedious to re-type large inputs every time a program runs. Reading from a file is the best way to consistently give the same input to a program repeatedly.

<fstream> is the library that provides access to File I/O in C++. It works so much like **<iostream>** that the same operators are used. The big difference is that file stream have to be explicitly opened by the coder, whereas **cout** and **cin** are opened for you every time you include **<iostream>**. Some tools provided by this library are:

- **ifstream**: an input file stream
- **ofstream**: an output file stream

```
ifstream studentFile;  
studentFile.open("gentry.txt");  
string fName, lName;  
float gpa;  
studentFile >> fName >> lName >> gpa;  
cout << lName << ", " << fName << ": " << gpa << endl;
```

Example 4: a code snip to read and print a file name gentry.txt



```
Atkinson, Gentry: 3.5  
  
Process returned 0 (0x0)   execution time : 0.002 s  
Press ENTER to continue.  
█
```

Example 5: sample output from Example 4

Expressions

Any combination of operators and operands in C++ is an expression. An operator is a built in tool used to manipulate data in a program. An operand is a piece of data that gets manipulated by an operator. An operand can be a single value ("literal"), a constant, or a variable ("mutable"). Some operators offered by C++ include:

- `+`, returns the sum of two operands
- `-`, returns the difference of two operands
- `*`, returns the product of two operands
- `/`, returns the dividend of two operands. Integer operands will return the result of *integer division*. If either operand is a float, the result will be a float.
- `%`, "modulus", returns the remainder of the division of two integers.
- `++`, "increment", increase the stored value of an operand by 1
- `--`, "decrement", decreases the stored value of an operand by 1
- `=`, "assignment", stores the value in the right in the variable on the left
- `==`, "equality comparison", returns true if the right and left hand operands are equal and false otherwise

Integer division can be difficult to understand. Try to recall being taught long division as an elementary school student. $10 / 3$ would have been computed as 3 remainder 1. The integer division of 10 by 3 is only 3. The modulus is 1. By contrast the floating point division of $10.0/3.0$ is 3.333... which is the answer most people would expect to see. If division is producing an unexpected answer in C++ be sure to check for integer division.

```
int a = 5, b = 6;
float c = 3.0, d = 10.0;

cout << "a/b = " << a/b << endl;
cout << "a/c = " << a/c << endl;
cout << "d/c = " << d/c << endl;
cout << "a*b = " << a*b << endl;
cout << "a*c = " << a*c << endl;
cout << "c*d = " << c*d << endl;
```

Example 6: some simple operations in C++

```
a/b = 0
a/c = 1.66667
d/c = 3.33333
a*b = 30
a*c = 15
c*d = 30

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Example 7: the output of Example 6

Operations are combined together to make statements. Every statement in C++ is terminated with a semicolon. Normally, each statement is written on its own line of the program but this is not required by C++. In fact, a whole program could be written on one line but it would be extremely difficult to read.

Order of operations matters when writing statements in C++. So:

$y = 5 * x + 3;$

is the same as:

$y = 3 + 5 * x;$

is the same as:

$y = 3 + (5 * x);$

Operators can be combined with the assignment operator to change a value and then store it in a variable. So:

$x += 5;$

is the same as:

$x = x + 5;$

Either one will store a value in the variable **x** that is 5 greater than its current value.