Strings and IO
Strings

- Integers and Characters
- Strings
- Input
- Formatted Output
Characters
Types – So Far

- To date all we have seen are numeric data types
  - int, short, long, long long, ...
  - float, double, ...
  - char
  - bool
- The char type stores a numeric code which is translated to a character when appropriate
  - And C++ treats them as numbers
Characters are Integers

- It’s easy to print the ASCII code for a character
  - `char ch = 'x';`
  - `cout << "code for " << ch;`
  - `cout << " = " << (int)ch;`
- The first `cout` statement prints the letter that the code represents
- The second `cout` statement prints the code
  - After first being cast (converted) to an `int`
- C++ will also allow arithmetic to be performed on char variables
  - The underlying numeric codes are operated on
Let’s say that we want to print all of the letters from A to Z

- We could write 26 `cout` statements
  
  ```cpp
  cout << 'A';
  cout << 'B';
  ...
  ```

- Or we could do this
  
  ```cpp
  char ch = 'A';
  while(ch < 'A' + 26){
    cout << ch << endl;
    ch++;}
  ```
Strings
Character Strings

- We have used (character) strings in `cout` calls
  - e.g. `cout << "Hello World";`
- It would also be useful to store strings in variables
  - And to get user input in the form of strings
- A string is a sequence of characters
  - Distinguished from an individual character
    - By being enclosed in ""s
C++ has two kinds of types

- Base types
  - int, char, float, bool, ...
- Structs and Classes
  - A variable of a class is referred to as an *object*
  - Objects may store many values and may have *methods*
  - A method is a function that belongs to a class

*string* objects store and manipulate strings

- The string class is contained in the string library
  - `#include <string>`
Let’s write a program to find out the name and age of the user
  - And then print them

We will store the name in a string object
  - Use `cin` to get the input for the name
  - And print it using `cout`
  - Both `cin` and `cout" know" what to do with string data
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string name;
    int age;

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

    cout << "Your name is " << name;
    cout << "", and your age is " << age;
    return 0;
}
String Methods

- The string class has a number of *methods*
  - A method is a function that belongs to an object variable
  - Object methods can be accessed using the member operator – a period (or dot, or full stop)
    - Known as dot notation
- We are not going to spend much time using methods in this course
  - But one example is the string *size* method
The structure that contains the characters in a string is an array

- An array is a sequence of variables of the same type
- Individual elements in an array can be accessed with a numerical index
  - Enclosed in []s following the name of the array

String objects allow the individual characters in the string to be accessed in this way
#include <iostream>
#include <string>
using namespace std;

const int ASCII_UC_LC = 32;

int main()
{
    string song = "Bat Out of Hell";
    string line = "The sirens are screaming, and the fires are howling";

    cout << "song length = " << song.size() << endl;
    cout << "line length = " << line.size() << endl;

    for (int i = 0; i < line.size(); ++i) {
        if (line[i] == 'e' || line[i] == 'g') {
            line[i] -= ASCII_UC_LC;
        }
    }

    cout << line;
    return 0;
}
Input Checking
As we’ve seen `cin` can be used to get input
  - It works appropriately regardless of the data type
    - Except that if you create your own classes `cin` will not magically know what to do with them
    - But you can give it this information
  - But entering data of an incorrect data type can result in errors
    - It turns out that input is relatively complex
      - Input is a common source of errors, and not just in C++
Basic input checking is relatively straightforward

- For example, where the user is expected to enter a number in a particular range

Basic idea

- Use a loop with a condition that checks that the value is within the desired range
- Prompt the user to enter a correct value in the loop body

But what happens if the user enters the wrong type?
#include <iostream>

using namespace std;

int getIntInRange(int low, int high)
{
    int result = low - 1;  // out of permitted range
    while (result < low || result > high) {
        cout << "Enter an int between " << low << " and " << high << ": " << endl;
        cin >> result;
    }
    return result;
}

int main()
{
    int x = getIntInRange(10, 100);
    cout << "You entered: " << x << endl;
    return 0;
}
Both `cin` and `cout` access what are referred to as streams

- A stream is a sequence of characters that are processed by `cin` or `cout`
  - For input or output

What happens when you enter keyboard data?

- The program doesn't begin processing the input until you press the `Enter` key
  - Though this behaviour can be changed
Consuming Stream Data

- Whatever is typed is put in an input stream
  - `cin` processes characters from the stream one at a time
  - If `cin` is reading data into an `int` variable it will keep reading integer characters from the stream
    - With an optional – (minus) character as the first character
    - Then the characters 0 to 9
- Any valid character is *consumed*
  - Removed from the stream
  - And `cin` only requests more data when the stream is empty
What happens when the user enters data that can not be processed by `cin`?

- Characters that are invalid for a variable
- `cin` fails
  - Nothing is read into the variable and
  - No characters are consumed from the stream
  - Our input function then tries again
    - But the stream still contains the same characters
    - `cin` attempts to insert them in the variable, and fails, ...
Once we know `cin` has failed we can fix the problem
- By throwing everything in the stream away
- And trying again
- But first we need to recognize that `cin` has failed
- Like a `string` variable `cin` is an object
  - Of the `istream` class
  - Which has its own methods
    - Functions that belong to it
  - One such function is `.fail()`
    - Which returns true if `cin` is in a failed state
There are two stages to fixing the input stream
  ▪ Clearing out the stream
  ▪ Resetting `cin` to a non failed state
  ▪ Use `clear` to reset the stream
    ▪ By calling `cin.clear()`
  ▪ Use `ignore` to clean out the stream
    ▪ `cin.ignore(10000, '\n');`
    ▪ Which removes the first $n$ characters, stopping at the first incidence of the second argument
      ▪ In the example: 10,000, ending when a newline is found
These are all reasonable questions. This method isn't going to work for every possible eventuality, but will deal with the majority of console input issues.
Be Careful

- Our function doesn't gracefully deal with every possible input problem
  - Even if we assume that the call to `ignore` is going to correctly handle clearing out the input stream
- It is usually good practice to clean out the input stream after each call to `cin`
  - In case the stream contains unwanted characters
```cpp
int main()
{
    int x = getIntInRange(10, 100);
    cout << "You entered: " << x << endl;
    int y = getIntInRange(3, 7);
    cout << "You entered: " << y << endl;
    return 0;
}
```

This is a slightly different example that uses the same input function twice.

Let's see what happens if the user decides to enter something odd.

A reasonable question might be why did the user type 23cats?

That doesn't really matter, as eventually some user will do something unexpected.

Notice that this is not a huge problem as it does not make the program crash (or go into an infinite loop) but it is definitely unattractive.

We can fix this by cleaning out the input stream at the end of the `getIntInRange` function.

23cats
int getIntInRange(int low, int high)
{
    int result = low - 1; // out of permitted range
    while (result < low || result > high) {
        if (cin.fail()) {
            cin.clear();
            cin.ignore(10000, '\n');
        }
        cout << "Enter an int between " << low << " and " << high << ": ";
        cin >> result;
    }
    cin.ignore(10000, '\n'); // clear input stream
    return result;
}

Once the user has entered an appropriate value the function clears out the input stream so that it is empty and ready for its next use
Formatting Output
We have been using `cout` to print program output

- Such output is not always formatted very attractively
  - Floating point numbers are printed to a default number of decimal spaces
  - Columns of output may not line up correctly

There are a number of formatting options

- Contained in the `<iomanip>` library
Setting Precision

- Precision sets the number significant digits
  - Can be changed by calling `setprecision(n)`
    - Where `n` is the number of significant digits to be displayed
  - The function is called in `cout`
    - `cout << setprecision(3);`
  - If the number of digits to the right of the decimal point is to be fixed
    - Use both `setprecision` and `fixed`

Example:
```
cout << setprecision(2) << 3.14159;  // displays: 3.1 – 2 significant digits
```
Using Fixed Floating Point Notation

- Fixed notation displays values without an exponent
  - And with the number of digits to the right of the decimal point set by the precision value
- To display numbers with two decimal places
  - `cout << fixed << setprecision(2);`
- There are other formatting options
  - Such as `scientific`
Setting Field Width

- It can be useful to set the output field width
  - So that output can be printed in columns, with values in rows lining up in columns
- The `setw(n)` function sets the field width
  - Where `n` is the width of the field in characters
  - Field width only applies to the output that immediately follows the `setw` call
    - Output is right justified within the field
Setting the Fill Character

- When `setw` is called output is right justified within the output field
  - By default the empty space is filled with spaces
    - So that it appears blank
  - The character that the empty space is filled with can be changed
  - By calling `setfill(ch)`
    - Where `ch` is a character
Write a program to print information about a loan and its compound interest

- Column heading should be underlined
- Values should be to two decimal places
- Values in columns should line up correctly