Objects as Arguments

ALWAYS
pass objects by reference!

(this includes strings!)

Why only pass objects by reference?

Passing by value creates a copy of the value being passed

- for primitives (int, bool, double, and char), this is no problem (8 bytes at most)
- an array variable only contains the *address* of the first element (4 bytes)
- copying an object, on the other hand, copies everything that comprises that object

Take a string variable, for example…

- a string containing "Hi, there" would be no problem to copy (still only about 15 bytes)
- a string containing the entire contents of the book *War and Peace* would require an additional 3,353,562 bytes (more than 3 *megabytes*) to copy!
- not only that, but the CPU will also be burdened with the work of needlessly duplicating all that data
- so, always pass objects by reference

I/O Stream Objects as Arguments

Here's an example of passing objects by reference:

```
void my_getline(istream& in, string& str, const char end) {
    char c;
    while (in >> c) {       // read 1 char at a time
        if (c == end) { // if the char matches the delimiter
            return;         // then we're all done
        }
        str += c;           // otherwise, append c to the string
    }
}
```

You can use <u>ANY</u> istream object with this function (cin, data files...)

I/O Stream Objects as Arguments

Here's a better example (make sure you understand why):

```
istream& my_getline(istream& in, string& str, const char end) {
    char c;
    while (in >> c) {       // read 1 char at a time
        if (c == endchar) { // if the char matches the delimiter
            break;          // then we're all done
        }
        str += c;           // otherwise, append c to the string
    }
    return in; // return the stream object
}
```

In this case, the istream object is also returned by reference!

- this is encouraged for stream objects (it allows I/O chaining and error checking)

I/O Stream Objects as Arguments

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

istream& my_getline(istream& in, string& str, const char end) {
    char c;
    while (in >> noskipws >> c) {
        if (c == end) {      // if the char matches the delimiter
            break;           // then we're all done
        }
        str += c;            // otherwise, append c to the string
        cout << str << endl;
    }
    return in; // return the stream object (allows chaining)
}

int main() {
    string text;
    ifstream infile("somefile.txt");

    // use my_getline() with cin...
    my_getline(cin, text, '\n');

    // use it with infile, too! ANY istream object will work
    my_getline(infile, text, '\n');

    return 0;
}
```

I/O Stream Objects as Arguments

Another example!

```
istream& read_data(istream& in, int data[], int& size) {
    int count = 0; // count how many values are successfully read
    in >> size;    // determine the required number of values
    while (count < size && in >> data[count]) {
        count++;
    }
    if (count != size) {
        cout << “Couldn’t read ” << size << “ values!” << endl;
    }
    return in; // return the stream object
}
```

I/O Stream Objects as Arguments

Look closely at the function header:

```
istream& read_data(istream& in, int data[], int& size);
```

return by reference!

The first argument is an istream object *by reference*

- remember, always pass objects by reference!

The function returns the exact same istream object, *also by reference*

- this is done by appending the ampersand to the function's return type
- without that ampersand, a copy of the object is made and then returned.
- we can check the returned stream object for failure, or we can chain input statements
- we've done this with other people's functions (`getline()`), but now we know how to do it ourselves.

I/O Stream Objects as Arguments

Here's an example that does output:

```
ostream& print_array(ostream& out, const int data[], int size) {
    for (int i = 0; i < size; i++) {
        out << data[i] << endl; // print each value to 'out'
    }
    return out; // return the stream object
}
```

You can use this function to print to <u>any</u> ostream object!

```
print_array(cout, numbers, num_values);    // print to cout
print_array(outfile, numbers, num_values); // print to a file
```

ALWAYS
pass objects by reference!

(this includes strings!)