Passing Streams to Functions

download the slides used in this presentation
PowerPoint pptx | Acrobat pdf

Objectives

While engaging with this module, you will...

1. learn how to pass file and output streams to and from functions
2. create new and efficient methods for sending information in-flow and out-flow around your program

Introduction

Stream objects can be passed to functions like any other kind of object. The hitch is that stream parameters must always be reference parameters. Let's take a look at two examples. The first is standard. The second will be your introduction to the concept of operator overloading.

A template function to open/check a file stream

```
// Pre: template parameter T must be either ifstream or ofstream type.
template <typename T>
void fileopen (T & filein, const string promptpart)
{
    const int MAX_TRIES = 5;
    int count = 0;
    string filename;
    cout<<"enter name of "<<promptpart<<" file: ";
    cin>>filename;
    filein.open(filename.c_str());

    while (!in)
    {
        in.clear(); // may be necessary on your platform
        cout<<"ERROR: file not connected. Try again……."<<endl;
        cout<<"enter name of "<<promptpart<<" file: ";
        cin>>filename;
        filein.open(filename.c_str());
        count++;

        if (count > MAX_TRIES)
        {
            cout<<"NOT CONNECTING AFTER "<<MAX_TRIES<<" ATTEMPTS…BAILING OUT" <<"……"<<endl;
            exit(1);
        }
    }

    return;
}
```

To call this function, you would send it an ifstream object and the constant literal "input" or an ofstream object and the constant literal "output". Notice the stream is passed by reference.

We will now see how the insertion operator, <<, is overloaded. This means that we are going to define how it should work for a user-defined type. I will show you the code here and then explain.

```
ostream& operator << (ostream & out, const point & p)
{
    out<<"("<<p.m_X<<", "<<p.m_Y<<")";
    return out;
}
```

The easy part: the body of the function is merely output formatting for the point struct we constructed in an earlier example. In order to fully explain the return type and parameter list, let's take a look at how it will be used.

```
cout<<p1;
or
cout<<p1<<" "<<p2;
or
ofstream fout;
...
fout<<p1<<endl;
```

Ok, so what is so profound here? Consider the first usage. The insertion operator, <<, is a binary operator, taking two operands. The operand on the left must be an ostream object. Ostream is a parent class of the ofstream class of objects. This means that any object that is of type ofstream, is also of type ostream. However, objects of type ostream, are not necessarily of type ofstream. cout is an ostream type object, but not an ofstream type object; it is an output stream, but not an output file stream. In my example, fout is an ofstream (output file stream) type object, and thus an ostream (output stream) type object. (This is all part of the topic called inheritance, of which you may learn much more in later coursework.) Thus, all the example usages above make sense. Now, the operand on the right must be a point type object. That is the type of object that we are wanting to stream (to the screen or a file). It is a const parameter because there is no reason for it to change, and reference to save copy overhead. If not reference, then time and memory is expended copying it. Many times, user-defined types are large, and you should make a habit of passing them as reference. Lastly, why do we return an ostream reference object? Again consider the first example. If the function were void, this would work fine. But then, look at the other examples. We want to be able to chain the function calls – output one point, then another, then a literal, then an endl, etc. If we return the very same ostream object passed, we accomplish this task.

```
cout<<p1 returns cout
    ->cout<<" " returns cout
        ->cout<<p2 returns cout
            ->cout is then ignored by the compiler
```

We will discuss overloading operators in much more detail later in the course.

One Final Note

As stated earlier, every ifstream object is an istream object, and every ofstream object is an ostream object. This implies that all those character input manipulation functions that we learned are available to cin and cout are also available to input file streams and output file streams! So,

```
in.get(char_var);
```

would retrieve data char-by-char from a file. What you can do with cin and cout you can do with file streams. The `getline()`, `ignore()`, `get()`, `putback()`, etc functions all work with file streams.