COMP322: Assignment 3 - Winter 2010

Due at 11:59pm EST, 24 Mar 2010

1 Introduction

In this assignment, we'll implement a simple class hierarchy, illustrating inheritance, virtual functions, plus a little bit of operator overloading and memory management.

In our little calculator program, we currently recognize variables and constants, both of which support only floating-point values.

Let's suppose we want to expand the language to support a wider range of types. In particular, we might want to add strings, functions, and arrays to our language. One way to do this would be to expand the `Symbol` and `Symtable` classes to store a object of a more complex class `Value` rather than a simple double-precision number.

Don't worry, we're not going to actually implement all of this machinery, we're just going to create the infrastructure that would make it possible!

2 Requirements

Create a class hierarchy as follows. Our base class will be the abstract class `Value`. We'll use this to derive three classes `NumericValue`, `StringValue`, and `ArrayValue`. The first two are straightforward, but the third is a bit tricky. We'll limit the complexity by implementing one-dimensional arrays with a fixed maximum size. However, these arrays may contain an arbitrary value in each location, *including another array*! Array accesses that are uninitialized, or beyond the maximum size, should return the numeric value zero.

Each derived class must provide a default constructor, plus a copy constructor and a destructor in the case of `ArrayValue`), in order to correctly handle memory management.

The two generic pieces of functionality your value class must support are a `print()` method, which will print the value to a given `ostream` object, and and a `clone()` method, which will make a copy of a `Value` object.

We've provided three files: The `Value.h` file defines all of the interfaces for your class hierarchy, a `a3main.cpp` file to test your class hierarchy, and `a3out.txt`, a sample output file which shows how the output of the `main()` program should appear. Your job is to create the `Value.cpp` file that actually implements the details of the methods.

3 Hints

Here is the definition of the abstract `Value` class from the header file:

```
class Value {
public:
  virtual ~Value();
  virtual void print(ostream &os) const = 0; // Print my value to 'os'
  virtual Value *clone() const = 0; // Make a copy of myself
};
```

Note that you should NOT modify `Value.h` or the `main()` program.

The `print()` method will be used to implement an overloaded << operator for the generic `Value` base class.

The `clone()` method serves a slightly less obvious function. It's needed to allow the `ArrayValue` class to correctly make private copies of any `Value` that is stored in an array. You'll want to use it in your copy constructor and in the `set()` method.

You'll need to initialize the static member `undefined` in the ArrayValue class. That will just be a line in your `Value.cpp` that looks like this:

```
NumericValue ArrayValue::undefined(0);
```

4 Submission

Please email your completed `Value.cpp` file to me via email. Good luck! Contact me or your T.A. with any questions or problems