C++ Interlude 6
Overloading and Friends

Pearson

Overloading C++ Operators

```
#include "BoxInterface.h"

template<class ItemType>
class PlainBox : public BoxInterface<ItemType>
{
private:
    ItemType  item;

public:
    PlainBox();
    PlainBox(const ItemType& theItem);
    virtual void setItem(const ItemType& theItem);
    virtual ItemType getItem() const;

  bool operator<(const
         PlainBox<ItemType>& rightHandSide) const;

};

#include "PlainBox.cpp"
#endif
```

Declare the overload in the header file.

PlainBox.h

```
// Create and initialize an array of boxes
const int NUM_BOXES = 5;
PlainBox<std::string> myBoxes[NUM_BOXES];
myBoxes[0] = PlainBox<std::string>("ring");
myBoxes[1] = PlainBox<std::string>("hat");
myBoxes[2] = PlainBox<std::string>("shirt");
myBoxes[3] = PlainBox<std::string>("sock");
myBoxes[4] = PlainBox<std::string>("shoe");
/
P
f
{
   if  (foundBox < myBoxes[i])
   {
      foundBox = myBoxes[i];
   }
}

std::cout << "Last item is : " << foundBox.getItem();
```

```
if ( foundBox.operator<(myBoxes[i]) )
```

Client Code

Overloading C++ Operators

```
#include "BoxInterface.h"

template<class ItemType>
class PlainBox : public BoxInterface<ItemType>
{
private:
    ItemType   item;

public:
    PlainBox();
    PlainBox(const ItemType& theItem);
    virtual void setItem(const ItemType& theItem);
    virtual ItemType getItem() const;

  bool operator<(const
        PlainBox<ItemType>& rightHandSide) const;

};

#include "PlainBox.cpp"
#endif
```

Declare the the overload in the header file.

`PlainBox.h`

```
#include "PlainBox.h"
template<class ItemType>
PlainBox<ItemType>::PlainBox()
{   } // end default constructor

template<class ItemType>
PlainBox<ItemType>::PlainBox(const ItemType& theItem)
{   item = theItem;  } // end constructor

template<class ItemType>
void PlainBox<ItemType>::setItem(const ItemType& theItem)
{  item = theItem;    } // end setItem

template<class ItemType>
ItemType PlainBox<ItemType>::getItem() const
{  return item;      } // end getItem

template<class ItemType>
```

```
    return item < rightHandSide.item;
```

```
{
   return item < rightHandSide.getItem();
}
```

Implement operation in source file.

`PlainBox.cpp`

OVERLOADING C++ OPERATORS

```
#include "BoxInterface.h"

template<class ItemType>
class PlainBox : public BoxInterface<ItemType>
{
private:
    ItemType   item;

public:
    PlainBox();
    PlainBox(const ItemType& theItem);
    virtual void setItem(const ItemType& theItem);
    virtual ItemType getItem() const;

 bool operator<(const
        PlainBox<ItemType>& rightHandSide) const;

  template<class friendItemType>
  friend std::ostream& operator<<(std::ostream& outStream,
        const PlainBox<friendItemType>& outputBox);

};

#include "PlainBox.cpp"
#endif
```

PlainBox.h

```
// Create and initialize an array of boxes
const int NUM_BOXES = 5;
PlainBox<std::string> myBoxes[NUM_BOXES];
myBoxes[0] = PlainBox<std::string>("ring");
myBoxes[1] = PlainBox<std::string>("hat");
myBoxes[2] = PlainBox<std::string>("shirt");
myBoxes[3] = PlainBox<std::string>("sock");
myBoxes[4] = PlainBox<std::string>("shoe");
 // Find box with last item alphabetically
 PlainBox<std::string> foundBox = myBoxes[0];
 for (int i = 1; i < NUM_BOXES; i++)
 {
    if (foundBox < myBoxes[i])
    {
      foundBox = myBoxes[i];
    }
 }
  std::cout << "Last item is : " << foundBox;
 cout << "Last item is : " << foundBox.getItem() << endl;
```

Client Code

OVERLOADING C++ OPERATORS

```cpp
#include "BoxInterface.h"

template<class ItemType>
class PlainBox : public BoxInterface<ItemType>
{
private:
    ItemType item;

public:
    PlainBox();
    PlainBox(const ItemType& theItem);
    virtual void setItem(const ItemType& theItem);
    virtual ItemType getItem() const;
 bool operator<(const
       PlainBox<ItemType>& rightHandSide) const;

  template<class friendItemType>
  friend std::ostream& operator<<(std::ostream& outStream,
       const PlainBox<friendItemType>& outputBox);
};

#include "PlainBox.cpp"
#endif
```

PlainBox.h

```cpp
template<class ItemType>
bool PlainBox<ItemType>::operator<(const
         PlainBox<ItemType>& rightHandSide) const
{
  return item < rightHandSide.getItem();
}


template<class friendItemType>
std::ostream& operator<<(std::ostream& outStream,
          const PlainBox<friendItemType>& outputBox);
{
  outStream << outputBox.item;
  return outStream;
}
```

No access modifier
No class namespace indicator
Stream fields to output stream
Return output stream

PlainBox.cpp

OVERLOADING C++ OPERATORS

```cpp
#include "BoxInterface.h"

template<class ItemType>
class PlainBox : public BoxInterface<ItemType>
{
private:
    ItemType item;

public:
    PlainBox();
    PlainBox(const ItemType& theItem);
    virtual void setItem(const ItemType& theItem);
    virtual ItemType getItem() const;
 b bool operator<(const
        PlainBox<ItemType>& rightHandSide) const;

  template<class friendItemType>
  friend std::ostream& operator<<(std::ostream& outStream,
       const PlainBox<friendItemType>& outputBox);

};
#include "PlainBox.cpp"
#endif
```

PlainBox.h

```cpp
// Create and initialize an array of boxes
const int NUM_BOXES = 5;
PlainBox<std::string> myBoxes[NUM_BOXES];
myBoxes[0] = PlainBox<std::string>("ring");
myBoxes[1] = PlainBox<std::string>("hat");
myBoxes[2] = PlainBox<std::string>("shirt");
myBoxes[3] = PlainBox<std::string>("sock");
myBoxes[4] = PlainBox<std::string>("shoe");
// Find box with last item alphabetically
PlainBox<std::string> foundBox = myBoxes[0];
for (int i = 1; i < NUM_BOXES; i++)
{
    if (foundBox < myBoxes[i])
    {
       foundBox = myBoxes[i];
    }
}

std::cout << “Last item is : ” << foundBox;
```

Client Code