G52CPP
C++ Programming
Lecture 22

Dr Jason Atkin

Dangers of arrays

And of STL containers

Same problems apply

Reminder: using **vector** example

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

int main()
{
   vector<char> v(10);
   // 10 elements

   int size = v.size();

   cout << "Size " << size
        << endl;

   // Set each value
   for( int i=0 ; i < size ; i++ )
       v[i] = i;

   // Iterate through vector
   vector<char>::iterator p
              = v.begin();

   for( ; p != v.end() ; p++ )
       *p += 97;

   // Output the contents
   for( int i=0 ; i < size ; i++ )
       cout << v[i] << endl;

   return 0;
}
```

A base class and sub-class

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

class Base
{
public:
   Base( int i = 1 )
      : i( i )
      {}

   virtual void out()
   {
      cout << "Base" << i << endl;
   }

protected:
   int i;
};
```

```
class Sub : public Base
{
public:
   Sub( int i = 2 )
      : i( i )
      , Base( i + 2 )
      {
      }

   void out()
   {
      cout << "Sub" << i << endl;
   }

protected:
   int i;
};
```

Simple code : create an object and use it

```
int main()
{
        Sub s( 3 );
        s.out();
        s.Base::out();

        Base& br = s;
        br.out();

        Base b = s;
        b.out();
```

- What is the output?

```
class Base
{
public:
   Base( int i = 1 )
      : i( i ) {}
   virtual void out()
   {cout << "Base" << i << endl;}
protected:
   int i;
};

class Sub : public Base
{
public:
   Sub( int i = 2 )
      : i( i ), Base( i + 2 ) {}
   void out() { … "Sub" … i }
protected:
   int i;
};
```

Simple code : create an object and use it

```
int main()
{
        Sub s( 3 );
        s.out();
        s.Base::out();

        Base& br = s;
        br.out();

        Base b = s;
        b.out();
```

```
class Base
{
public:
   Bas

   vir
   {co                                  endl;}
protec
   int
};

class
{
public
   Sub
                                    2 ) {}
   void out() { … "Sub" … i }
protected:
   int i;
};
```

Sub3
Base5

Sub3

Base5

- What is the output?

Creating an array : what is the output?

```
vector<Sub> v1( 10 );
for ( int i = 0;
      i < v1.size(); i++ )
{
   Sub s( i );
   v1[i] = s;
}

// Output the contents
vector<Sub>::iterator p1 =
      v1.begin();
for ( ; p1 != v1.end(); p1++ )
   p1->out();

for ( int i = 0;
      i < v1.size(); i++ )
   v1[i].out();
```

```
class Base
{
public:
   Base( int i = 1 )
      : i( i ) {}
   virtual void out()
   {cout << "Base" << i << endl;}
protected:
   int i;
};

class Sub : public Base
{
public:
   Sub( int i = 2 )
      : i( i ), Base( i + 2 ) {}
   void out() { … "Sub" … i }
protected:
   int i;
};
```

Creating an array : what is the output?

```
vector<Sub> v1( 10 );
for ( int i = 0;
      i < v1.size(); i++ )
{
   Sub s( i );
   v1[i] = s;
}

// Output the contents
vector<Sub>::iterator p1 =
      v1.begin();
for ( ; p1 != v1.end(); p1++ )
   p1->out();

for ( int i = 0;
      i < v1.size(); i++ )
   v1[i].out();
```

```
class Base
{
public:
   Base( int i = 1 )
      :
   virt
   {cou                                  << endl;}
protect
   int
};

class S
{
public:
   Sub(
      :                                  + 2 ) {}
   void                                  … i }
protected:
   int i;
};
```

Sub0
Sub1
Sub2
Sub3
Sub4
Sub5
Sub6
Sub7
Sub8
Sub9

Creating an array of base class objects

```
vector<Base> v2( 10 );
for ( int i = 0;
      i < v2.size(); i++ )
{
   Sub s( i );
   v2[i] = s;
}

// Output the contents
vector<Base>::iterator p2 =
      v2.begin();
for ( ; p2 != v2.end(); p2++ )
   p2->out();

for ( int i = 0;
      i < v2.size(); i++ )
   v2[i].out();
```

```
class Base
{
public:
   Base( int i = 1 )
      : i( i ) {}
   virtual void out()
   {cout << "Base" << i << endl;}
protected:
   int i;
};

class Sub : public Base
{
public:
   Sub( int i = 2 )
      : i( i ), Base( i + 2 ) {}
   void out() { … "Sub" … i }
protected:
   int i;
};
```

Creating an array of base class objects

```
vector<Base> v2( 10 );
for ( int i = 0;
      i < v2.size(); i++ )
{
   Sub s( i );
   v2[i] = s;
}

// Output the contents
vector<Base>::iterator p2 =
      v2.begin();
for ( ; p2 != v2.end(); p2++ )
   p2->out();

for ( int i = 0;
      i < v2.size(); i++ )
   v2[i].out();
```

```
class Base
{
public:
   Base( int i = 1 )
      :
   virt
   {cou                                  << endl;}
protect
   int
};

class S
{
public:
   Sub(
      :                                  + 2 ) {}
   void                                  … i }
protected:
   int i;
};
```

Base2
Base3
Base4
Base5
Base6
Base7
Base8
Base9
Base10
Base11

Creating array of base class pointers

```
vector<Base*> v3( 10 );

// Set each value
for ( int i = 0;
      i < v3.size(); i++ )
{
   Sub s( i );
   v3[i] = &s;
}

// Iterate through vector
vector<Base*>::iterator p3 =
      v3.begin();

for ( ; p3 != v3.end(); p3++ )
   (*p3)->out();
```

```
class Base
{
public:
   Base( int i = 1 )
      : i( i ) {}
   virtual void out()
   {cout << "Base" << i << endl;}
protected:
   int i;
};

class Sub : public Base
{
public:
   Sub( int i = 2 )
      : i( i ), Base( i + 2 ) {}
   void out() { … "Sub" … i }
protected:
   int i;
};
```

Creating array of base class pointers

```
vector<Base*> v3( 10 );

// Set each value
for ( int i = 0;
      i < v3.size(); i++ )
{
   Sub s( i );
   v3[i] = &s;
}

// Iterate through vector
vector<Base*>::iterator p3 =
      v3.begin();

for ( ; p3 != v3.end(); p3++ )
   (*p3)->out();
```

```
class Base
{
public:
   Base( int i = 1 )
      :
   virt
   {cou                                  << endl;}
protect
   int
};

class S
{
public:
   Sub(
      :                                  + 2 ) {}
   void                                  … i }
protected:
   int i;
};
```

```
Sub9
Sub9
Sub9
Sub9
Sub9
Sub9
Sub9
Sub9
Sub9
Sub9
```

Creating array of base class pointers

```
vector<Base*> v4( 10 );

// Set each value
for ( int i = 0;
      i < v4.size(); i++ )
{
   v4[i] = new Sub(i);
}

// Iterate through vector
vector<Base*>::iterator p4 =
      v4.begin();

for ( ; p4 != v4.end(); p4++ )
   (*p4)->out();
```

```
class Base
{
public:
   Base( int i = 1 )
      : i( i ) {}
   virtual void out()
   {cout << "Base" << i << endl;}
protected:
   int i;
};

class Sub : public Base
{
public:
   Sub( int i = 2 )
      : i( i ), Base( i + 2 ) {}
   void out() { … "Sub" … i }
protected:
   int i;
};
```

Creating array of base class pointers

```
vector<Base*> v4( 10 );

// Set each value
for ( int i = 0;
      i < v4.size(); i++ )
{
   v4[i] = new Sub(i);
}

// Iterate through vector
vector<Base*>::iterator p4 =
      v4.begin();

for ( ; p4 != v4.end(); p4++ )
   (*p4)->out();
```

Warning: we didn't delete the objects which we created!

```
class Base
{
public:
   Base( int i = 1 )
      :
   virt
   {cou                                  << endl;}
protect
   int
};

class S
{
public:
   Sub(
      :                                  + 2 ) {}
   void                                  … i }
protected:
   int i;
};
```

Sub0
Sub1
Sub2
Sub3
Sub4
Sub5
Sub6
Sub7
Sub8
Sub9

Similarly: arrays of pointers

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

class Base
{
public:
  Base() : i(1) {}
  virtual void out()
    { cout <<"Base" << i <<endl; }
  int i;
};

class Sub : public Base
{
public:
  Sub() { i = 2; }
  void out()
    {cout <<"Sub" << i <<endl; }
};
```

```
int main()
{
  Sub* arrayp1[3] = { new Sub,
        new Sub, new Sub };
  Base* arrayp2[3] = { new Base,
        new Base, new Base };
  // Output Array 1
  for ( int i = 0 ; i < 3 ; i++ )
    arrayp1[i]->out();
  // Output Array 2
  for ( int i = 0 ; i < 3 ; i++ )
    arrayp2[i]->out();
  // Copy Array 1 elements to 2
  for ( int i = 0 ; i < 3 ; i++ )
    arrayp2[i] = arrayp1[i];
  // Output Array 2
  for ( int i = 0 ; i < 3 ; i++ )
    arrayp2[i]->out();
}
```

Arrays of pointers

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

class Base
{
public:
  Base() : i(1) {}
  virtual void out()
    { cout <<"Base" <<i <<endl; }
  int i;
};

class Sub : public Base
{
public:
  Sub() { i = 2; }
  void out()
    {cout <<"Sub" <<i <<endl; }
};
```

```
int main()
{
  Sub* arrayp1[3] = { new Sub,
        new Sub, new Sub };
  Base* arrayp2[3] = { new Base,
        new Base, new Base };
  // Output Array 1
  for ( int i = 0 ; i < 3 ; i++ )
```

Call out() on each object in arrayp1

```
  // Output Array 2
  for ( int i = 0 ; i < 3 ; i++ )
```

Call out() on each object in arrayp2

```
  // Copy Array 1 elements to 2
```

Copy subclass pointers to Base* array

```
  arrayp2[1] = arrayp1[1];
  // Output Array 2
```

Call out() on each object in arrayp2
i.e. the Base* array, with Sub* objects

```
}
```

Objects are not pointers

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

class Base
{
public:
  Base() : i(1) {}
  virtual void out()
    { cout <<"Base" <<i <<endl; }
  int i;
};

class Sub : public Base
{
public:
  Sub() { i = 2; }
  void out()
    {cout <<"Sub" <<i <<endl; }
};
```

```
int main()
{
Sub array1[3];
Base array2[3];

// Output Array 1
for ( int i = 0 ; i < 3 ; i++ )
  array1[i].out();
// Output Array 2
for ( int i = 0 ; i < 3 ; i++ )
  array2[i].out();
// Copy array 1 to 2
for ( int i = 0 ; i < 3 ; i++ )
  array2[i] = array1[i];
// Output Array 2
for ( int i = 0 ; i < 3 ; i++ )
  array2[i].out();
}
```

Arrays of objects

Objects are not pointers

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

class Base
{
public:
  Base() : i(1) {}
  virtual void out()
    { cout <<"Base" <<i <<endl; }
  int i;
};

class Sub : public Base
{
public:
  Sub() { i = 2; }
  void out()
    {cout <<"Sub" <<i <<endl; }
};
```

```
int main()
{
Sub array1[3];
Base array2[3];

// Output Array 1
f
  array1[1].out();
// Output Array 2
  array2[1].out();
// Copy array 1 to 2
  array2[1] = array1[1];
// Output Array 2
f
}
```

Arrays of objects

Call out() on each object in arrayp1

Call out() on each object in arrayp2

Copy subclass OBJECTS to Base array

Call out() on each object in array2
i.e. the Base array, after Sub copy

The slicing problem

The slicing problem

- In these examples, when we:
  - Stored base class objects
  - Assign sub-class objects only to them
- Only the base class part was stored

- We sliced off the sub-class part

- This may be obvious, but things can get worse…

What is the slicing problem?

- Passing an object type parameter **by value** uses the copy constructor – i.e. copies it
- Assigning an object to another object copies the values, using the assignment operator
- If the thing you are copying to is a base class object ***(or thinks it is!)*** the base class assignment operator is used
- **Neither the default copy constructor nor assignment operator are virtual**
  - The base class version gets used!!!
  - Just making them virtual would not help anyway
- The slicing problem occurs when you treat a sub-class as the base class for a copy/assignment
  - Only the base class parts get copied
  - i.e. the sub-class parts are sliced off

How can it happen?

- By using references or pointers, e.g.:

```
class BaseClass
{
public:
   int MyParam;
};

class SubClass
   : public BaseClass
{
public:
   int MySubClassParam;
};
```

```
int main()
{
   SubClass s1, s2;

   BaseClass& rs1 = s1;
   BaseClass& rs2 = s2;
   rs1 = rs2;

   BaseClass* ps1 = &s1;
   BaseClass* ps2 = &s2;
   *ps1 = *ps2;
}
```

- The sub-class part will not be copied

Why?

- The slicing problem occurs when you treat a sub-class as the base class for a copy/assignment
  - Only the base class parts get copied
  - i.e. the sub-class parts are sliced off
- A function like this was created and used:

```
Base& operator=(const Base& rhs )
{
        this->MyParam = rhs.MyParam;
        return *this;
}
```

Note:
Different return types!
Different parameter types!

- Rather than using the sub-class version:

```
SubClass& operator=(const SubClass& rhs )
{
        this->MySubClassParam = rhs.MySubClassParam;
        this->MyParam = rhs.MyParam;
        return *this;
}
```

Advanced stuff

Advanced ‘stuff’

- Thing to remember:
  - Use assignment with base class pointers and references with care (or not at all)
  - You may end up slicing off the sub-class part
- But some of this you ***could*** actually fix, if you REALLY wanted to…
  - Probably too much work to do so usually
- The following slides are complicated stuff, just to show you what you COULD do if you really wanted to, and give you a taste of the power of operator overloading…

1: virtual assignment operator

- **First make the assignment operator in the base class `virtual`**
  - **I.e. implement it yourself so that you can make it virtual**

```
virtual Base& operator=(const Base& rhs )
{
   this->i = rhs.i;
   return *this;
}
```

2: Sub-class assignment operator

- **Add assignment operator in the sub-class**
- **Note: This is NOT an overload of the base class operator= since it takes a sub-class object reference**

```
Sub& operator=(const Sub& rhs )
{
   this->Base::operator=(rhs);
   this->j = rhs.j;
   cout << "copied sub" << endl;
   return *this;
}
```

Use scoping to call base class version, not the sub-class one (virtual function!)

3: Override base class version

- **In the SUB CLASS also provide an overload of the BASE CLASS assignment operator, which checks for sub-class objects and if so calls the sub-class assignment operator**

```
virtual Base& operator=(const Base& rhs )
{
   try
   { // If object really is a sub (sub-class object)
       const Sub& rsub = dynamic_cast<const Sub&>(rhs);
       *this = rsub; // Assign using 2 subclass objs
   }
   catch( bad_cast )
   {   // Object is NOT a sub (sub-class object)
       this->Base::operator=(rhs); // Use base class
   }
   return *this;
}
```

Next lecture

- New C++11 features it is useful to know about

- Functors and lambda functions – very useful and increasingly important