Sizeof and Vtables

ITP 435 – Spring 2016
Week 2, Lecture 1
Lecturer: Sanjay Madhav

USC Viterbi
School of Engineering

University of Southern California

sizeof Operator

Tells you the number of bytes a particular data type (or variable) takes up:

```
// Returns 4
sizeof(int)

char a;
// Returns 1
sizeof(a)

// Returns 1
sizeof(bool)
```

sizeof Question #1


```
int* ptr;
ptr = new int[10];
```

Q: What is the `sizeof(ptr)`??

A: On a 32-bit system, 4. On a 64-bit system, 8.

sizeof Question #2


```
int array[10];
```

Q: What is the `sizeof(array)`??

A: 40, because each `int` is 4 bytes big.

sizeof Question #3


```
int array[10];
int* ptr = array;
```

Q: What is the `sizeof(ptr)`??

A: It's a pointer, so you'll get either 4 or 8 again!

sizeof Question #4


```
class Test1
{
      char c;
      int i;
};
```

Q: What is the `sizeof(Test1)` ??

A: 8, because of padding.

Class Padding

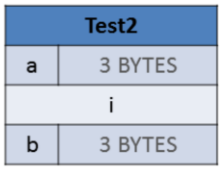
- By default, Visual Studio (and most compilers) guarantees alignment to be equal to the size of the element
- ***x-byte aligned*** means the variable's memory address is guaranteed to be divisible by x
- So our previous example, we have a 4 byte variable (`int i`) that must be 4-byte aligned. Thus, we need padding:

Test1	
c	3 BYTES
i	

More Padding

What happens with this?

```
class Test2
{
      char a;
      int i;
      char b;
};
```



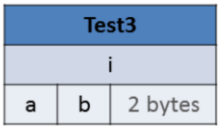

```
sizeof(Test2) == 12
```

The compiler won't rearrange variables!!

Even More Padding

Solution: Always declare your variables from largest to smallest:

```
class Test3
{
      int i;
      char a;
      char b;
};
```



USC Viterbi School of Engineering

University of Southern California

Since a/b are one byte in size, they don't have to guarantee any alignment.

There still is 2 bytes at the end because we have to guarantee alignment for i in the case of an array of Test3 classes

bool bit trick


```
// sizeof(Test4) is 12
class Test4
{
     int i;
     bool a, b, c, d, e, f, g;
};

// sizeof(Test5) is 8
// a:1 tells Visual Studio to use only one bit for the variable
class Test5
{
     int i;
     bool a:1, b:1, c:1, d:1, e:1, f:1, g:1;
};
```

sizeof Question #5

Q: What's the sizeof this class?

```
class VirtualClass
{
      int i;
      virtual void F();
};
```

A: 8 (for a 32-bit program). Because it has a virtual function, the size of the class is the int + the pointer to a virtual function table.

Cost of Virtual Functions

Memory cost:

- Need to add 1 pointer to the start of a class' data once it has virtual functions.
- This points to the "virtual function table" or vtable.

Performance cost:

- At run-time, when a virtual function is called virtually, the vtable pointer is dereferenced, which then finds the correct function to call from the vtable.

Virtual Function Tables

- Simple example:

```
class A
{
public:
   virtual void a();
   virtual void b();
   virtual ~A();
};

class B : public A
{
public:
   virtual void a();
   virtual void c();
   virtual ~B();
};
```

vtable for A (12 bytes if 32-bit)	
Index	**Function Pointer**
0	&A::a()
1	&A::b()
2	&A::~A()

vtable for B (16 bytes if 32-bit)	
Index	**Function Pointer**
0	&B::a()
1	&A::b()
2	&B::~B()
3	&B::c()

USC Viterbi School of Engineering — University of Southern California

B's vtable must use the indices established by A, and only build on it. This is how at runtime if you call a() on a pointer to an A, it knows to call whatever function is pointed to by index 0.

A couple more sizeof/vtable examples…


```
class X
{
   int a;
public:
   virtual void x();
   void y();
   virtual void z();
   virtual ~X();
};
```

```
sizeof(X) == 8
```

X
vtable*
a

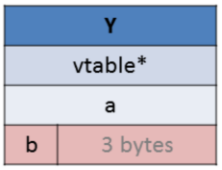
vtable for X (12 bytes if 32-bit)	
Index	**Function Pointer**
0	&X::x()
1	&X::z()
2	&X::~X()

Notice that the function y is not in the vtable, because y is not a virtual function.

A couple more sizeof/vtable examples…


```
class Y : public X
{
   char b;
public:
   virtual void z() override;
};
```

`sizeof(Y) == 12`



vtable for Y (12 bytes if 32-bit)	
Index	**Function Pointer**
0	&X::x()
1	&Y::z()
2	&X::~X()

Notice that the function y is not in the vtable, because y is not a virtual function.

sizeof Question #6

Q: What's the sizeof this class?

```
class Empty
{
};
```

A: 1. Classes cannot take up 0 memory.

sizeof Question #7

Q: What is the sizeof the class "Derived"?

```
class Empty
{
};
class Derived : public Empty
{
     int i;
};
```

A: 4. Even though Empty has a sizeof 1, the compiler can optimize away that for Derived, since Derived has an int it wouldn't be empty.

This is called the "empty base optimization."