CS360

Windows Programming

Last Week

- We
  - Covered the principles behind web services
  - Introduced the .NET framework
  - Reviewed object-oriented programming
  - Started learning about C#

C# Statements and Expressions

- Very similar to C++, with some changes to increase robustness
  - No `‘->’` or `‘::’`; all qualifications use `‘.’`
  - Local variables must be initialized before use
  - if, while, do require bool condition
  - goto can’t jump into blocks
  - switch statement – no fall through (empty case OK)
  - case labels can be strings

Types

- FCL is a library of types
  - Classes – today
  - Structs – today
  - Interfaces – next time
  - Enumerations – next time
  - Delegates – next time

Classes

- Classes contain
  - Fields - data members in C++
  - Methods - member functions in C++
  - Properties - expose data using get and set
  - Events - define notifications that class can hire

Example

```
class Rectangle
{
    // Fields
    protected int width = 1;
    protected int height = 1;

    // Methods (constructors)
    public Rectangle () {}
    public Rectangle (int cx, int cy)
    {
        Width = cx;
        Height = cy;
    }
}
```

Example (cont.)

```
// Properties
   public int Width
   {
       get { return width; }
       set
       {
           if (value > 0)
               width = value;
           else
               throw new ArgumentOutOfRangeException (
                   "Width must be 1 or higher");
       }
   }
```

Example (cont.)

```
public int Height
    {
        get { return height; }
        set
        {
            if (value > 0)
                height = value;
            else
                throw new ArgumentOutOfRangeException (
                    "Height must be 1 or higher");
        }
    }

    public int Area
    {
        get { return width * height; }
    }
```

Example (cont.)

```
Rectangle rect = new Rectangle ();
Rectangle rect = new Rectangle (3, 4);

rect.Width *= 2;
int area = rect.Area;
```

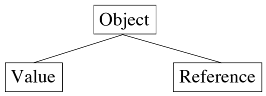
Structs

```
struct Point
{
 public int x;
 public int y;
 public Point (int x, int y)
 {
  this.x = x;
  this.y = y;
 }
}
```

Reference and Value Types

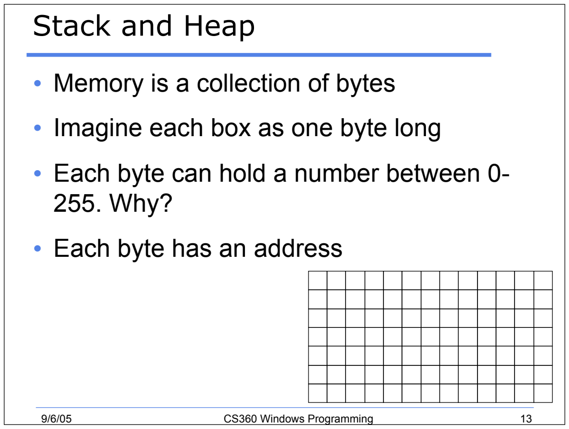
- Classes are reference types
- Structs are value types

Common Type System



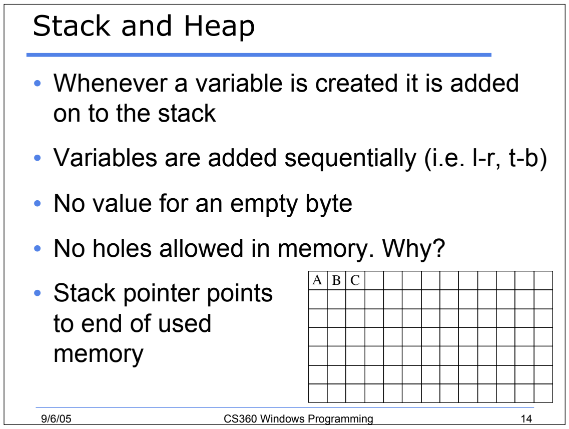
Built-in (primitive) data types
Examples: int, double, …
Size is specific
Allocated on Stack
Assignment copies value
Cannot derive from other types
User defined types: structs
Deallocated when defining block exits

Examples: Classes, Arrays, Interfaces, Delegates
Allocated on managed heap
Assignment copies reference
Garbage collected



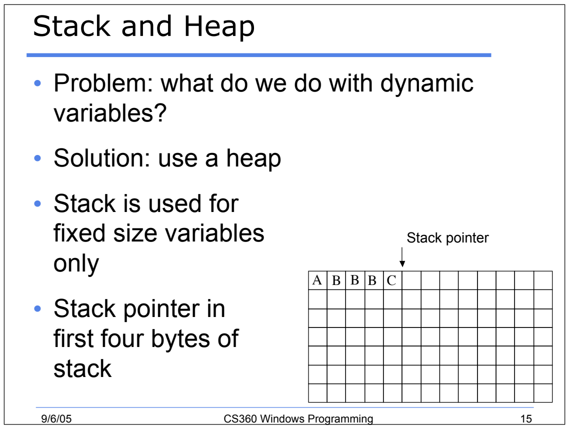
Stack and Heap

- Memory is a collection of bytes
- Imagine each box as one byte long
- Each byte can hold a number between 0-255. Why?
- Each byte has an address



Stack and Heap

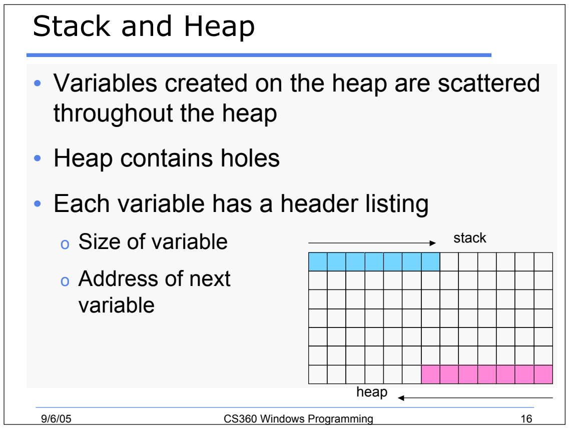
- Whenever a variable is created it is added on to the stack
- Variables are added sequentially (i.e. l-r, t-b)
- No value for an empty byte
- No holes allowed in memory. Why?
- Stack pointer points to end of used memory



Stack and Heap

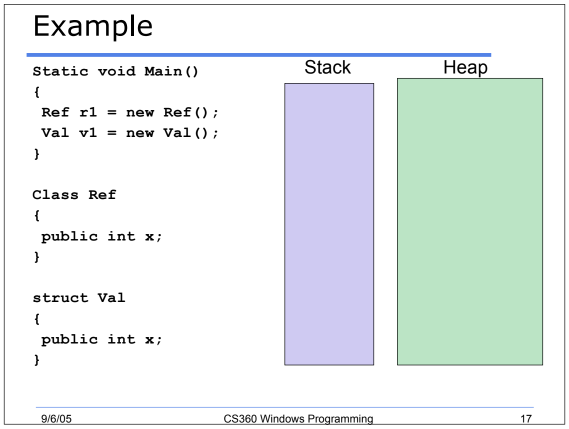
- Problem: what do we do with dynamic variables?
- Solution: use a heap
- Stack is used for fixed size variables only
- Stack pointer in first four bytes of stack

Stack pointer
A
B
B
B
C



Stack and Heap

- Variables created on the heap are scattered throughout the heap
- Heap contains holes
- Each variable has a header listing
  - Size of variable
  - Address of next variable

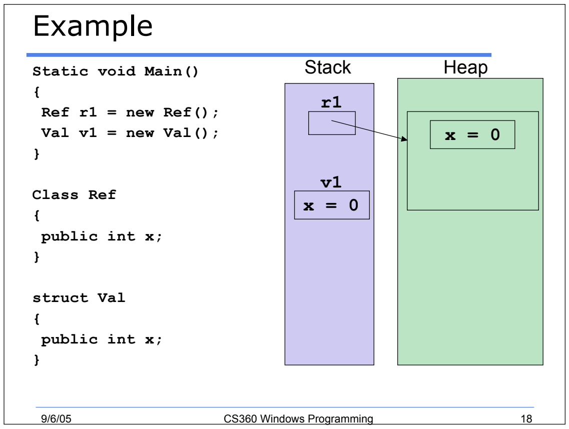


Example

```
Static void Main()
{
 Ref r1 = new Ref();
 Val v1 = new Val();
}

Class Ref
{
 public int x;
}

struct Val
{
 public int x;
}
```



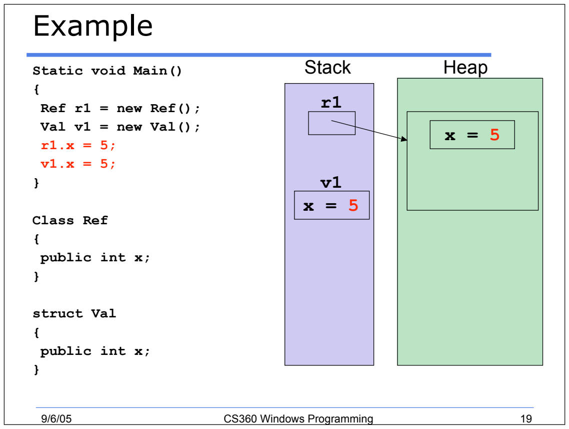
Example

```
Static void Main()
{
 Ref r1 = new Ref();
 Val v1 = new Val();
}

Class Ref
{
 public int x;
}

struct Val
{
 public int x;
}
```

Stack
Heap
r1
v1
x = 0
x = 0

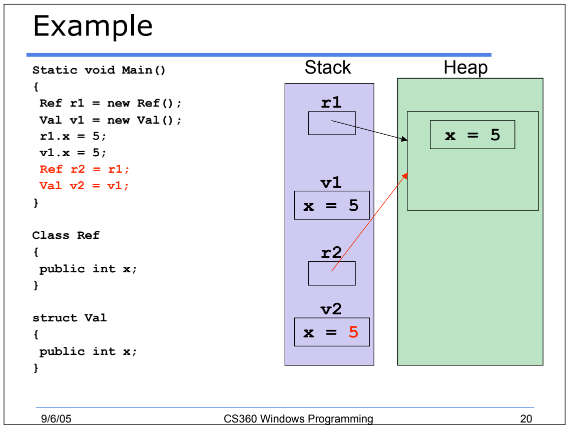


Example

```
Static void Main()
{
 Ref r1 = new Ref();
 Val v1 = new Val();
 r1.x = 5;
 v1.x = 5;
}

Class Ref
{
 public int x;
}

struct Val
{
 public int x;
}
```

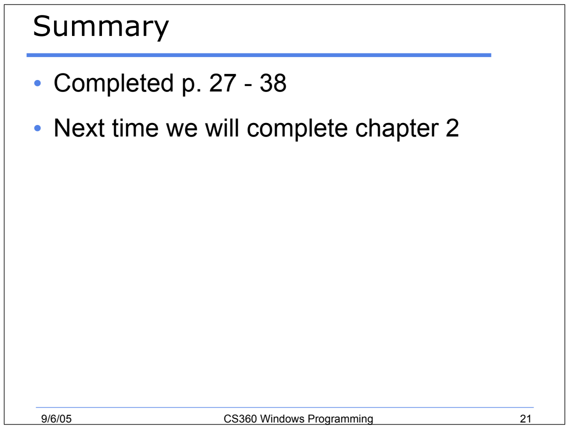


Example

```
Static void Main()
{
 Ref r1 = new Ref();
 Val v1 = new Val();
 r1.x = 5;
 v1.x = 5;
 Ref r2 = r1;
 Val v2 = v1;
}

Class Ref
{
 public int x;
}

struct Val
{
 public int x;
}
```



Summary

- Completed p. 27 - 38
- Next time we will complete chapter 2