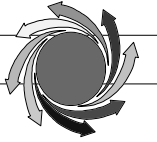


Dynamic Objects



- **So far . . .**
  - All objects defined at compile time
    - Except objects in STL containers?
- **What if . . .**
  - We don't know in advance . . .
  - What kind of objects or how many?
  - Objects created/destroyed at run time

---

---

---

---

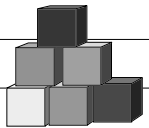
---

---

---

---

“Conventional” Objects



- **Static objects**
  - Fixed memory allocation at compile
  - In “static” area of memory
- **Automatic objects**
  - Memory allocated during function
  - In “stack” area of memory
    - More on stacks later (CS-285)

---

---

---

---

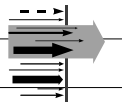
---

---

---

---

Dynamic Memory



- **Usage not known in advance**
  - Depends on actions of program
- **Draws from a “pool” (heap, free store)**
  - Memory allocated as needed
    - For dynamic objects
  - Released when no longer needed
    - Memory may be reused

---

---

---

---

---

---

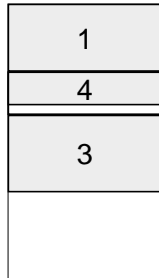
---

---

...

Heap Example

- Allocate block 1
- Allocate block 2
- Allocate block 3
- Release block 2
- Allocate block 4



Block 4 smaller than block 2

.....

---

---

---

---

---

---

---

---

...

Dynamic Object Use

- **To create object**
  - Allocate memory from heap
  - Initialize normally (constructor)
- **How do we reference?**
  - Dynamic object has no name
  - Access through a pointer object

.....

---

---

---

---

---

---

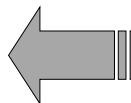
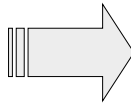
---

---

...

Dynamic Object Operators

- **Operator new**
  - Allocates memory for object
  - Invokes constructor
    - With or without arguments
  - Returns pointer to object
- **Operator delete**
  - Destroys object
  - Releases memory to heap



.....

---

---

---

---

---

---

---

---

...

**Object Creation Example**

```

Date* dp1; ← Define pointer objects
Date* dp2; ← Create dynamic objects

dp1 = new Date;
dp2 = new Date(1,5,2001);

*dp1 = *dp2; ← Access objects via pointers

cout << dp2->GetDay() << endl;

```

.....

---

---

---

---

---

---

---

---

...

**Pointer Assignment Revisited**

```

int* x = new int(3);
int* y = new int(5);

*x = 9;
x = y;
*x = 7;

```

Can the object at "2074" be accessed after "x=y"?

.....

---

---

---

---

---

---

---

---

...

**Memory Leaks**

- **Dynamic object allocated**
  - Associated with a pointer
- **But pointer is lost**
  - Pointer object set to different value
  - Pointer object lifetime ends
- **Dynamic object inaccessible!**

.....

---

---

---

---

---

---

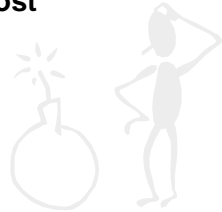
---

---

...

Memory Leak Problems

- **Object inaccessible**
  - May contain important information
- **Dynamic memory “lost”**
  - Has been allocated
    - Cannot be reused
  - No way to release it




---

---

---

---

---

---

---

---

...

Destroying Dynamic Objects

```
Date* dp;           Create dynamic object
dp = new Date(2,4,2005);
cout << dp->GetDay() << endl;

delete dp;         Destroy dynamic object
                   (call destructor); release
                   dynamic memory
```




---

---

---

---

---

---

---

---

...

What About This?

```
Date* dp;         Define pointer object

cout << dp->GetDay() << endl;

Access object via pointer (?)
```

What is the value of “dp”?  
 What object is it pointing at?




---

---

---

---

---

---

---

---

...

Pointer Initialization & Cleanup

```
Date* dp = 0; // Or NULL
dp = new Date(4,7,1776);
cout << dp->GetDay() << endl;
delete dp;
```

```
dp = 0;
```

Initialize to NULL if no object; reset to NULL after destroying object

.....

...

More on Null Pointers

- **Must not dereference**
  - Don't apply "\*" operator to null pointer
- **OK to apply delete operator**
  - Does nothing if pointer value is null
- **After delete ...**
  - Set pointer object to zero (NULL)
    - Unless pointer object is going away

.....

...

Null Pointers, Zero, and NULL

- **Null pointer value**
  - Special value meaning "no object"
    - May be binary zero (or maybe not!)
- **Zero**
  - Can be converted to null pointer
- **NULL**
  - Another name for zero (<cstdlib>)

.....

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---


---

---

---

...

### Testing Pointer Values



```

Date* dp = 0;
if (dp != 0)
  cout << dp->GetDay() << endl;
else
  cout << "Null pointer" << endl;

if (dp) cout << "Pointer OK";
  
```

Test pointer value

True if pointer not null

.....

---

---

---

---

---

---

---

---

...

### Dynamic Arrays

- **Normal arrays**
  - Size specified at compile time
  - E.g., Date holidays [20]
- **But what if we don't know?**
  - Needed size not known at compile
- **Allocate arrays at run time**
  - Similar to dynamic objects

.....




---

---

---

---

---

---

---

---

...

### Dynamic Array Creation

```

Date* pa1;
Date* pa2;

unsigned int n1 = 10;
pa1 = new Date [n1];
pa2 = new Date [n1 * 2];

pa1[3] = pa2[5];

*pa1 = *pa2;
  
```

Define pointer objects

Create dynamic arrays

Access elements

What about this?

.....

---

---

---

---

---

---

---

---

⋮

Array Element Initialization

- **Dynamic array creation**
  - E.g., `Date* pa = new Date[10];`
- **Each element is initialized**
  - By a `Date` constructor
- **But no constructor arguments**
- **Must have no-arg constructor**
  - Or dynamic array not possible!

\_\_\_\_\_ ⋮

---

---

---

---

---

---

---

---

⋮

Destroying Dynamic Arrays (1)

- **Create dynamic object**
  - E.g., `Date* pa = new Date;`
- **Create dynamic array**
  - E.g., `Date* pa = new Date[10];`
- **How to destroy?**
  - If "`delete pa;`", what happens?
    - Memory deallocation?
    - Destructor invocation?



\_\_\_\_\_ ⋮

---

---

---

---

---

---

---

---

⋮

Destroying Dynamic Arrays (2)

- **Pointer points to . . .**
  - A single object
  - Or an array of objects
- **Must tell compiler which it is**
  - Single object: `delete pa;`
  - Array of objects: `delete [] pa;`
  - Match `new` with `delete`!
    - Single object or array forms

\_\_\_\_\_ ⋮

---

---

---

---

---

---

---

---