Scope and Lifetime

- The *scope* of a variable is the environment in which the variable is visible
- The *lifetime* of a variable is the period of the execution during which the variable is defined
- Commonly the same

Automatic vs. Dynamic Objects

- Automatic variables “automatically” created by entering scope, and destroyed by exiting scope
- Dynamic variables require dynamic allocation and deletion

Dynamic Allocation

- Storage for object dynamically allocated
- Ex: `void main() { f(); }`
  `void f() {`
  `  Frame* x;`
  `  x = new Frame("Test");`
  `}
- Lifetime of dynamically allocated object determined by `new` and `delete`
Dynamic Data Allocation

- Dynamically allocated memory located on the “heap”
- Heap is block of memory devoted to dynamic allocation
- Operator `new` requests allocation of certain sized chunk
- Operator `delete` returns memory to heap

Typical Process Memory Layout

Typical Process Memory Layout

Working with Dynamic Objects

- Use `new` to create object
  ```
  Frame *window; //pointer
  window = new Frame("One",10,20,50,50);
  ```
- Manipulate through pointer
  ```
  window->MoveTo(50, 50);
  ```
- Destroy with `delete`
  ```
  delete window;
  ```

Using Delete

- Be careful to use `delete` in the same way you used `new`
- Example: array of ints
  ```
  int *iarray = new int[SIZE];
  delete[] iarray;
  ```
- Especially important if array is of objects for which destructor must be called

Problems

- Aliases – two pointers to the same object
  - Changes to one are changes to the other
  - Deleting one, invalidates other
- Memory leaks – pointer lifetime ends before memory deallocated
- Dangling pointers – memory deallocated before end of pointer lifetime

Aliases

- Requires assignment of one pointer variable to another
  - Copy constructors
  - Assignment operators
- Never allow aliases, unless going to manage with reference counting
Memory Leaks

- Source: forgetting to delete allocated memory
- Delete should occur in either
  - Procedure where variable declared
  - Destructor of class where allocated
- Be careful not to delete too soon, or too late (or never)!

Dangling Pointers

- Sources:
  - Deleting alias – remaining pointer points to reclaimed memory
  - Methods that return pointer to
    - Object local to method
    - Object internal to other object
- Similar problem when returning references

Example: A String Class

```cpp
class String {
   //partial decl
   public:
     String();
     String(const String&);
     String(const char*);
     String& operator=(const String&);
     ~String();
   private:
     char *rep;
}
```

Default Constructor

- Constructor responsible for allocating storage for pointer

```cpp
String::String() {
  rep = new char[1];
  rep[0] = '\0';
}
```

Assignment Operator

```cpp
String& String::operator=(const String& s){
  if (rep != s.rep) {
    delete[] rep;
    int s_length = s.length() +1;
    rep = new char[s_length];
    ::strcpy(rep,s.rep);
    return * this;
  }
}
```

Destructor

- Destructor responsible for cleaning up object when it is destroyed

```cpp
String::~String() {
  delete[] rep;
}
```