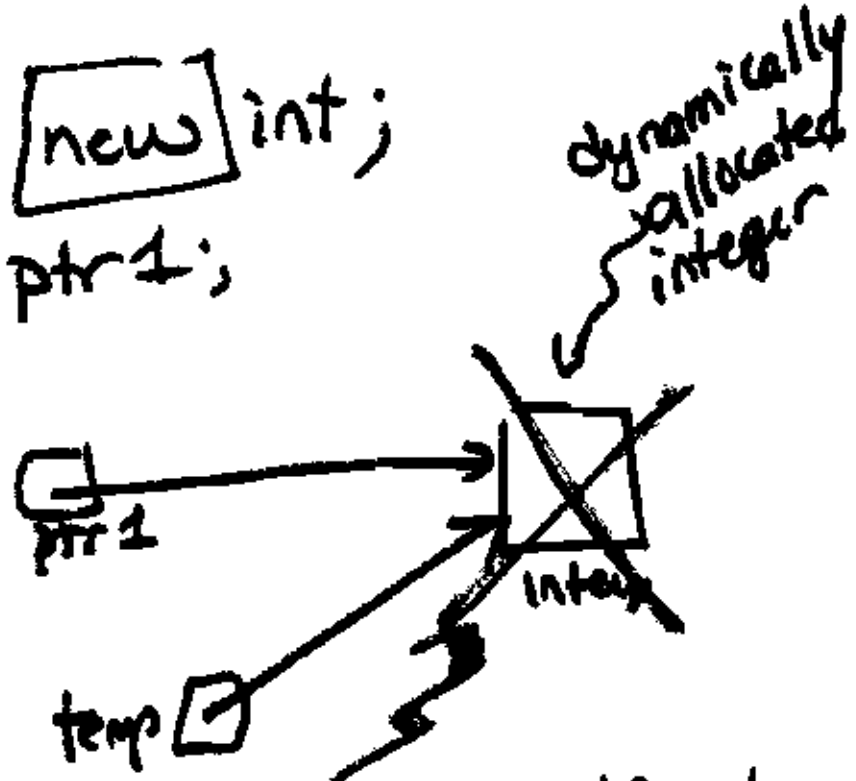
```
int *ptr1, *temp;
```

ONCE → `ptr1 = new int;`

`temp = ptr1;`




ONCE →

```
// when done
delete temp;
temp = NULL;
ptr1 = NULL;
```

← deallocates memory that temp is pointing to

# Arrays

Statically Allocated Array

"compile time"

char name[21];

constant

name

constant pointer to the first element (char)

cin >> name;

0

20

Dynamically Allocated Array

"run time"

char * array;

array = new char[7];

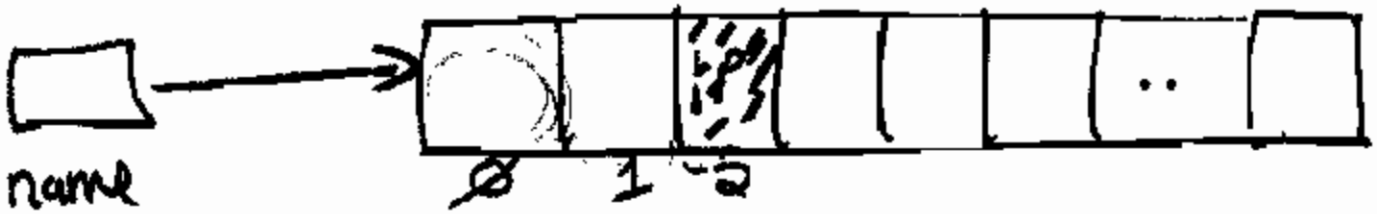
variable

cin >> array;

array

0

6




Contiguous

name[2] = 'p';

*(name+2) = 'p';

*(name + 2) Pointer ~~Cast~~ Arithmetic

pointer to char

2 * sizeof(char)
= bytes in a char

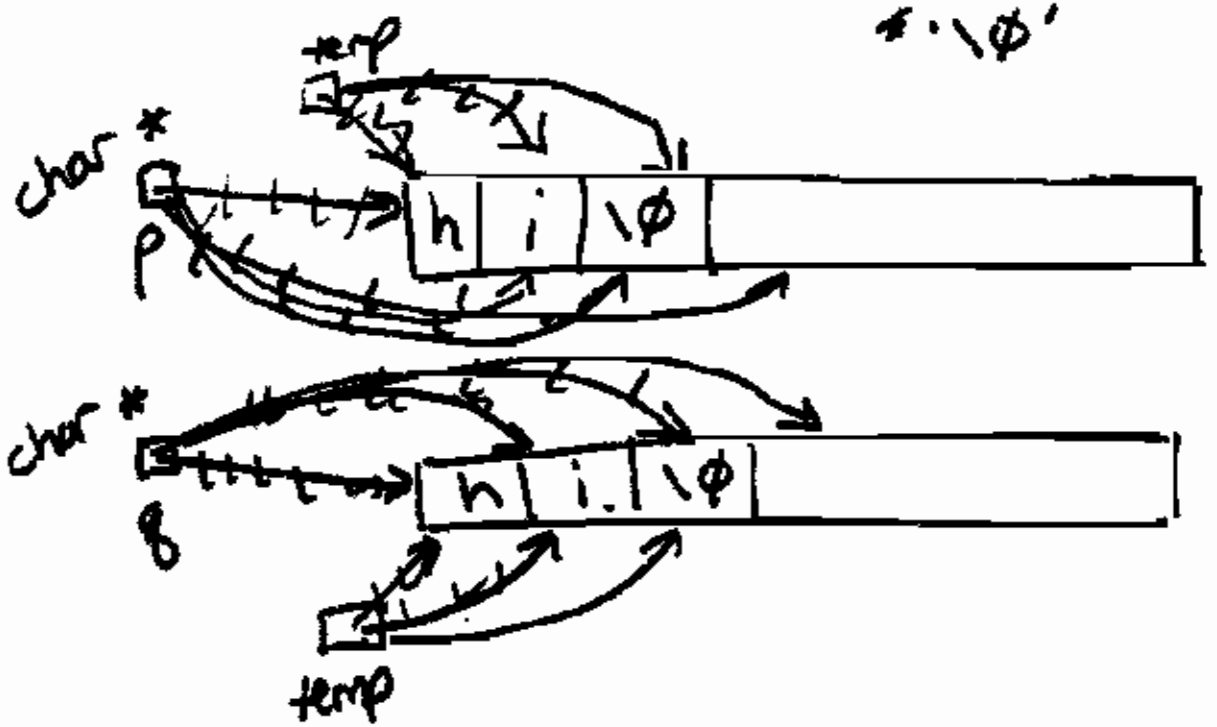
address

while ( *p++ = *q++ )
Assignment
NULL Body
p
q
test
test

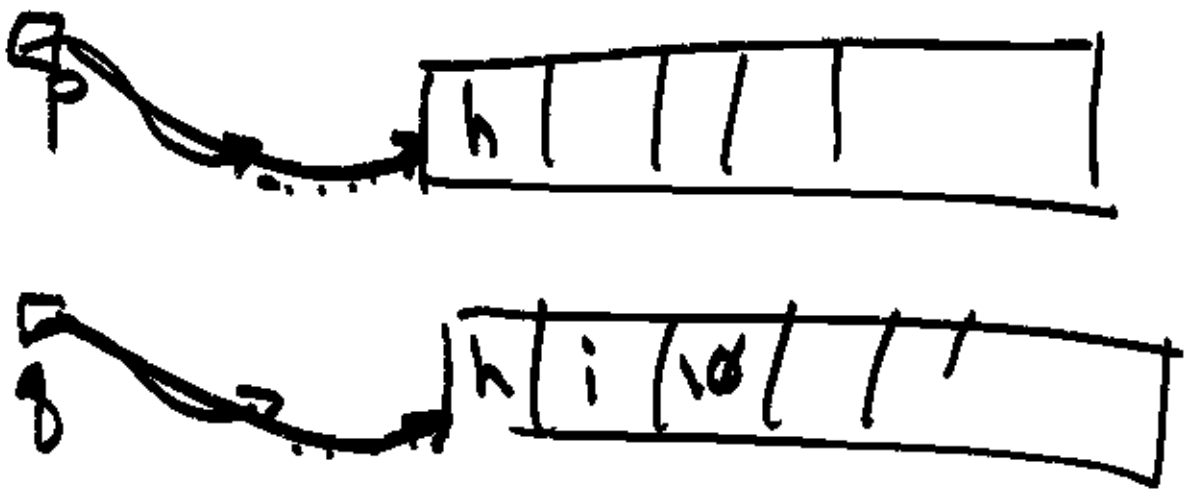
```
Strcpy ( char *p, char *q)
{
        while (*p++ = *q++) ;
}
```

*(q++)

'\0'



```
--q;
--p;
while (*(++p) = *(++q));
```



# Pointers and Structs

```
struct video
{
    char * name;
      :
};
```

char name[21];

an array of characters

```
video a_movie;
a_movie.name = new char[strlen(movie)+1];
strcpy (a_movie.name, movie);
```

in a function

OR

```
video library[100];
library[i].name = new char[strlen(movie)+1];
strcpy (library[i].name, movie);
```

later on:

whenever deallocating memory for an array

```
delete [] amovie.name;
```

---

or

```
for(int i=0; i<nummovies; ++i)
    delete [] library[i].name;
```

```
videolibrary::videolibrary()
{
    library = new video[SIZE];
    size_of_array = SIZE;
    num_videos = 0;
}
```

OR

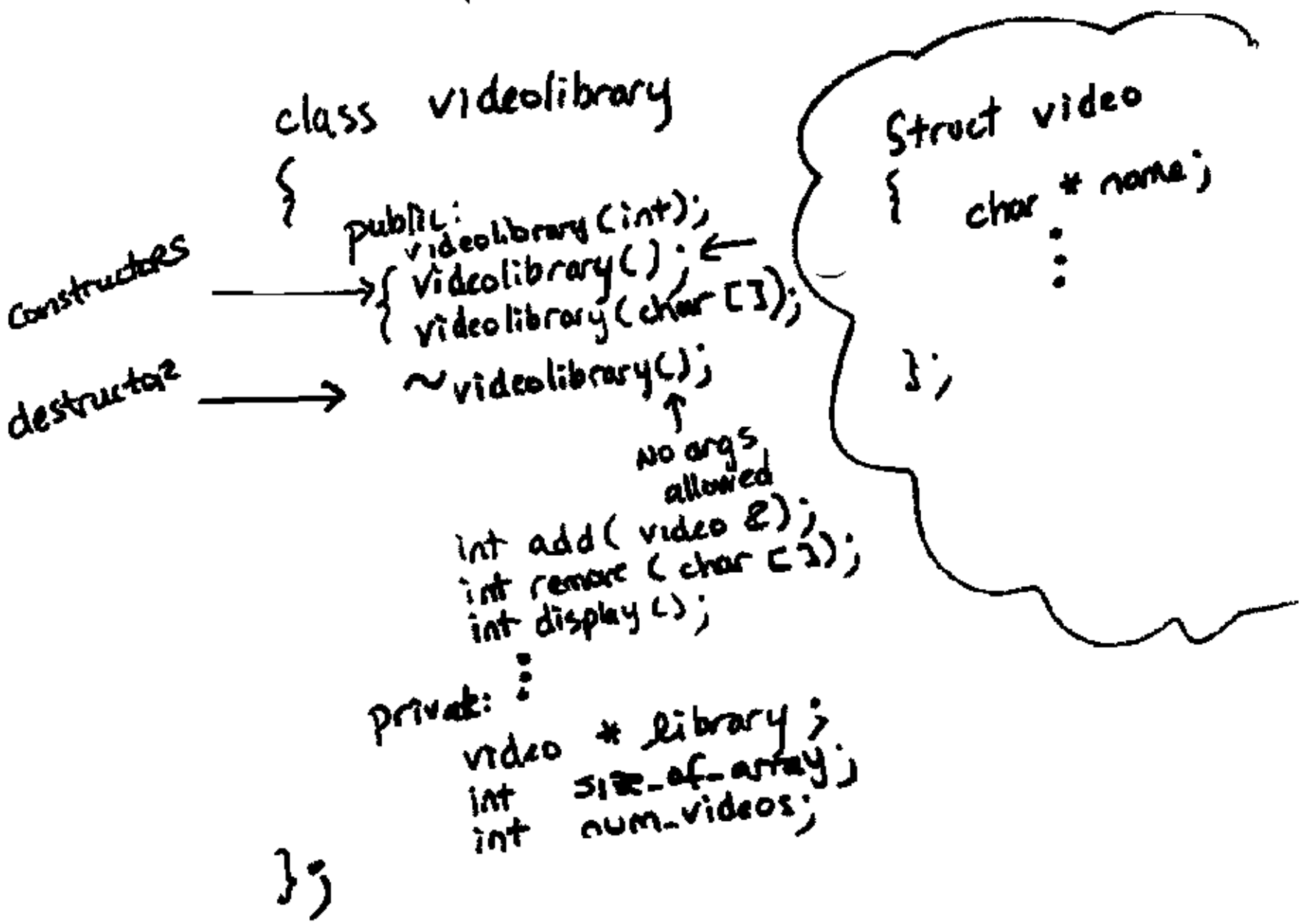
INSTEAD USE A "Default" ~~Constructor~~ WITH "DEFAULT" Args

```
videolibrary::videolibrary(int size)
{
    library = new video[size];
    size_of_array = size;
    num_videos = 0;
}
```

int size = SIZE in prototype

# classes & dynamic memory





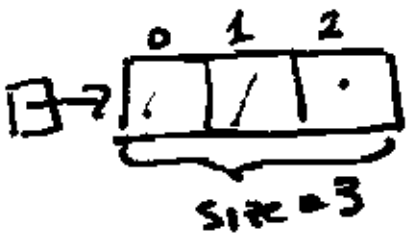
Prototype for default ARGs

public:

VideoLibrary(int size = SIZE);

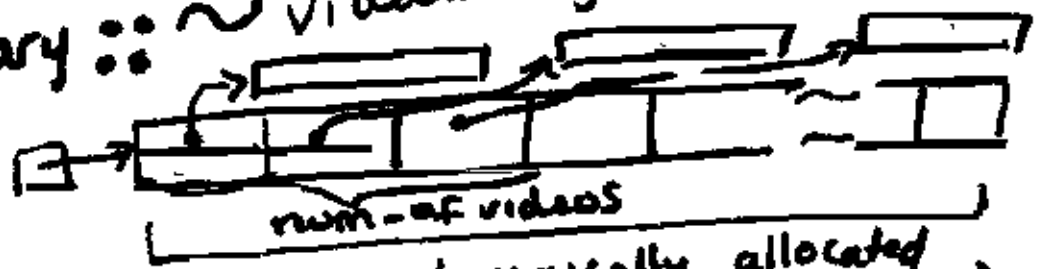
NOW **DO NOT** supply the regular "default" constructor

```
videolibrary::videolibrary(int size)
{
    library = new video[size];
                ↑ Do I want to pad this?
    size_of_array = size;
    num_videos = 0;
}
int videolibrary::display()
{
    for(int i = 0; i < num_videos; ++i)
        cout << library[i].name
             << endl;
    return num_videos;
}
int videolibrary::add(video &toadd)
{
    if(num_videos < size_of_array)
    {
        library[num_videos].name = new
            char[strlen(toadd.name)+1];
        strcpy(library[num_videos].name, toadd.name);
        ++num_videos;
```



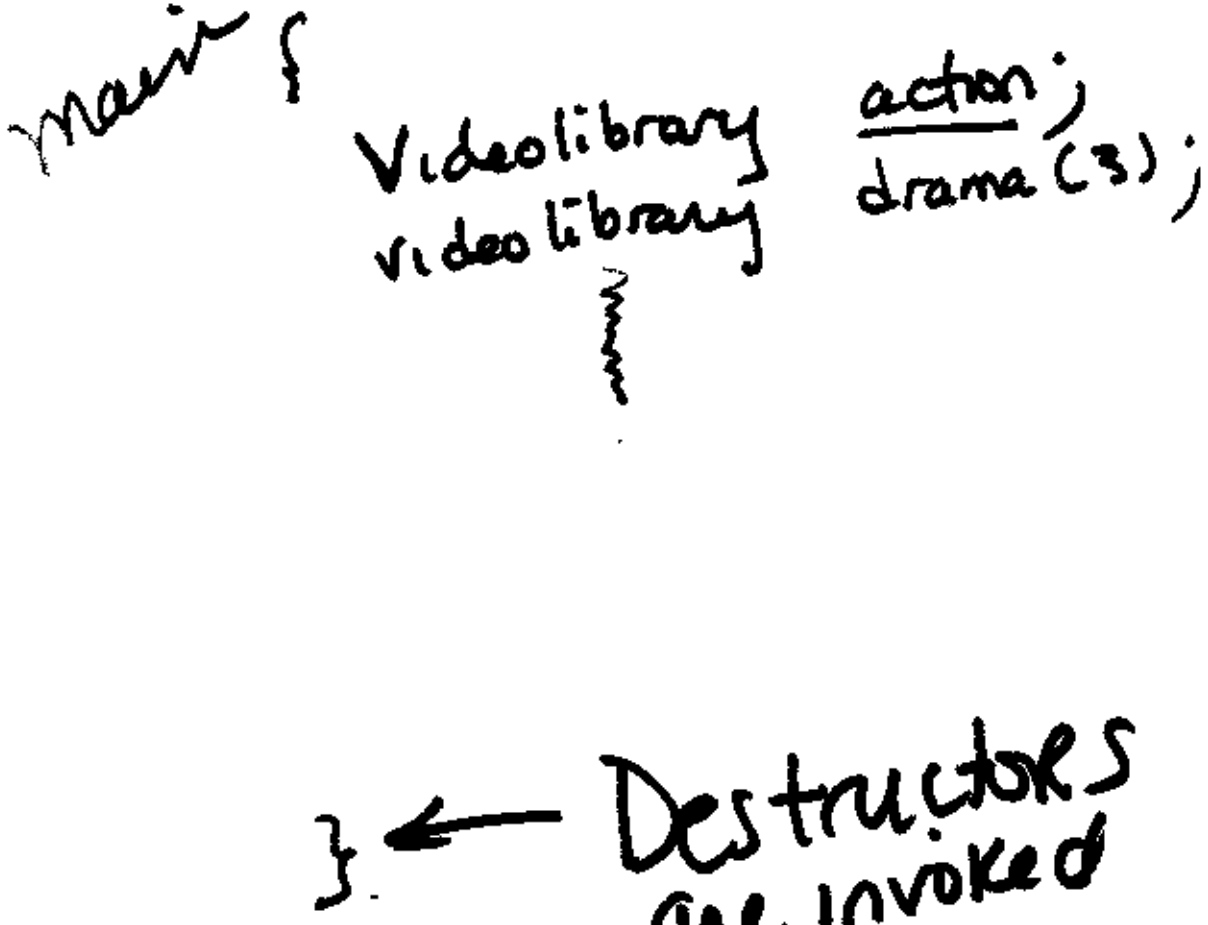

Destructor — Deallocate the dynamic Memory Managed by the class & do any other work to close down the object (e.g., write the array's data out to a file)

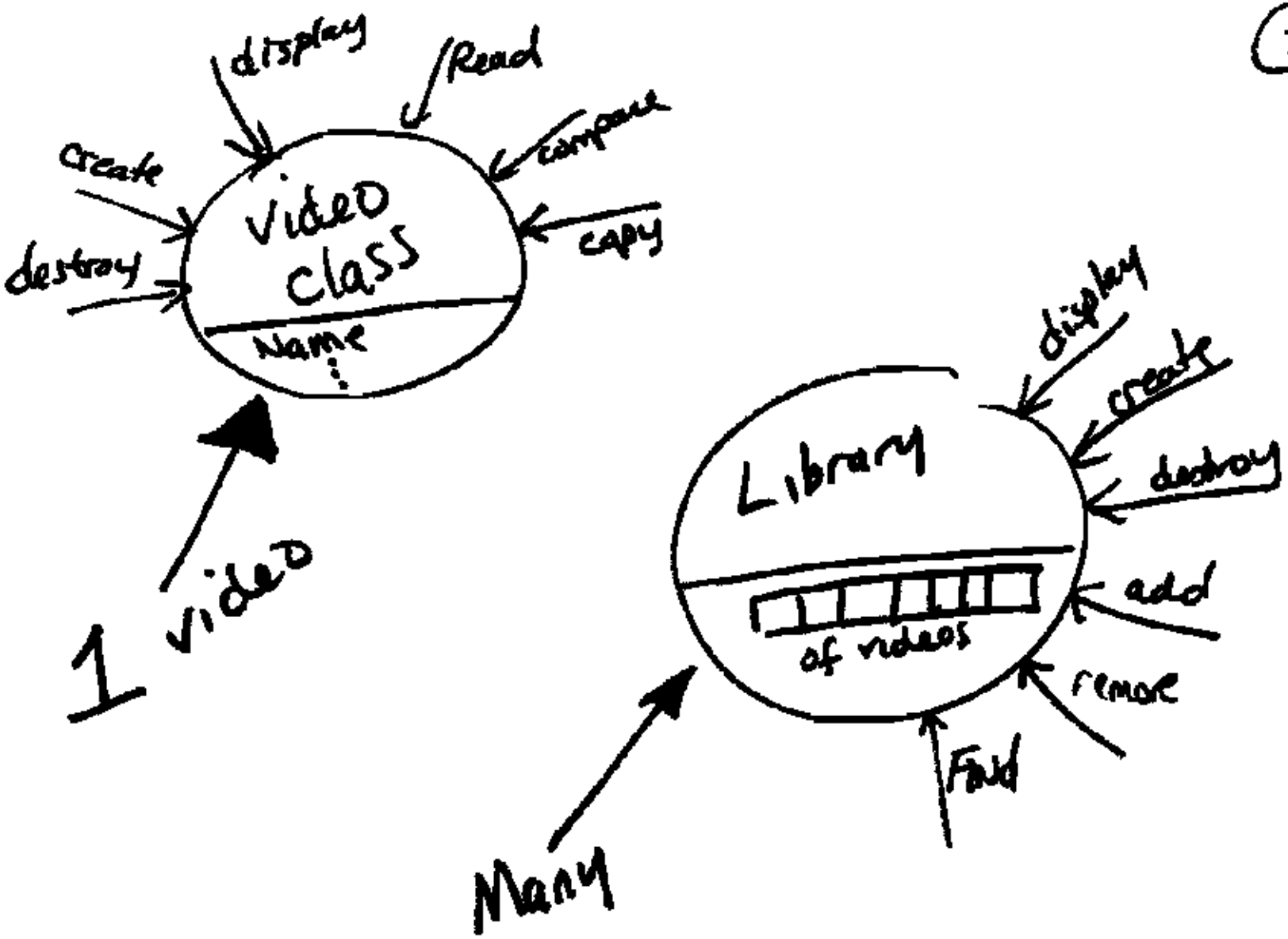
```
VideoLibrary::~VideoLibrary()
{
```




```
    for (int i = 0; i < num_of_videos; ++i)
        delete [] library[i].name;

    delete [] library;
    library = NULL;
}
```


main {
VideoLibrary videoLibrary;
action();
drama(3);
}
Destructors are invoked


display
Read
compare
create
Video
Class
copy
destroy
Name
1
Video
Library
display
create
destroy
add
of videos
remove
Find
Many

```
class video
{
    public:
        video();
        ~video();
        int read();
        int display();

    private:
        char * name;
        .
        .
        .
};
```

```
class videoLibrary
{
    public:
        videoLibrary();
        videoLibrary(int);
        ~videoLibrary();
        int add(); //read
        int display();

    private:
        ~~library~~ *
        video * library;
        int num_of_video;
        int size_of_array;
};
```

```
video::video()
{     name = NULL;
}
video::~video()
{
      delete [] name;
      name = NULL;

 }
int video :: read()
{  char temp[100];
   cout << "Enter a video";
   cin.get (temp, 100);
   cin.ignore(100, '\n');

   name = new char
     [strlen (temp) + 1];
   strcpy (name, temp);
        :
}
```

```
int video :: display()
{
    cout << name
         << endl;
    return 1;
}
```

(W)

```
int videolibrary :: add()
{  if (num_of_videos < size_of_array)
   {
      library[num_of_videos] .read();
   }  ++num_of_videos;
   return num_of_videos;
}
int videolibrary :: display()
{ for(int i=0; i<num_of_videos; ++i)

      library[i].display();

   return num_of_videos;
}
videolibrary :: ~videolibrary()
{     delete [] library;
      library = NULL;
}
```

a video

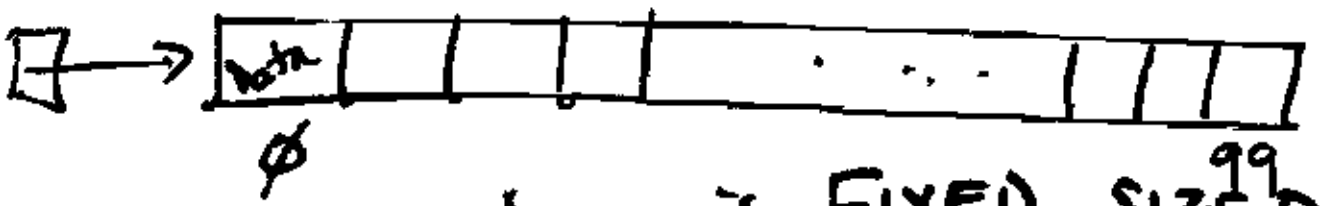
# Constructors

Main

```
VideoLibrary movies; //uses default constructor
VideoLibrary drama(10);
VideoLibrary action("action.dat");
VideoLibrary beachHouseMovies[100];
```

array of 100 libraries

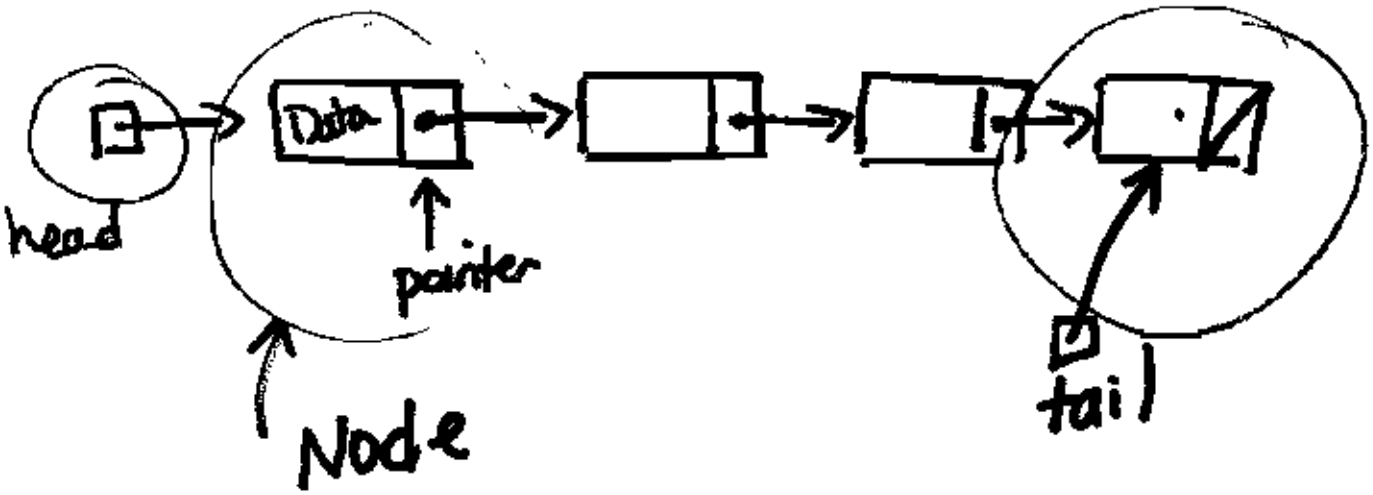
? default constructor will be called 100 times




Array is FIXED SIZED ENTITY

vs

Linear Linked List



Struct node


```
{
    video movie;
    node * next;      // where the next node exists
                      // Recursive definition
};
```

link

node * head;

head = NULL;

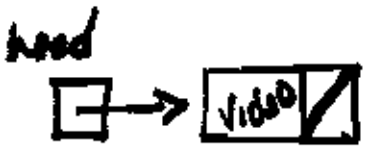
head

---

Add a video

head

"Before"




"After"