

# EECS402 Lecture 14

Andrew M. Morgan

Revisiting Use Of:  
Copy Constructors  
Assignment Operators  
Destructors

1



## Problem Description

EECS  
402

- Recall:
  - Copy ctors are used when a copy of an object needs to be made (most often via a pass-by-value parameter)
  - Assignment operators are used to assign one object's attributes to another object's
- Both of these do a "member-by-member copy"
  - Algorithm: For each data member of the right-hand-side object, copy its value into the corresponding member of the left-hand-side object
  - If the member is an integer, the 4 bytes containing the int are copied
  - If the member is a double, the 8 bytes containing the double are copied
  - If the member is a pointer, the 4\* bytes containing the pointer are copied
    - NOTE: Only the 4\* bytes of the pointer are copied, even if the pointer is pointing to an array of 100 values!

\*: These days, pointers are usually 8 bytes, but I still talk about them like they're 4 bytes since it makes it easier to include in examples showing memory, etc.

EECS  
402

Andrew M Morgan

2



2



## A Dynamic Array Class

EECS  
402

```
class DynArrayClass
{
private:
    int num;
    char *vals; //Will point to
                //array of chars
public:
    void setVals(char a, char b,
                 char c)
    {
        num = 3;
        vals = new char[num];
        vals[0] = a;
        vals[1] = b;
        vals[2] = c;
    }

    void setVals(char a, char b,
                 char c, char d)
    {
        num = 4;
        vals = new char[num];
        vals[0] = a;
        vals[1] = b;
        vals[2] = c;
        vals[3] = d;
    }

    void changeVal(int index, char c)
    {
        if (index >= 0 && index < num)
        {
            vals[index] = c;
        }
        else
        {
            cout << "Out of range" << endl;
        }
    }

    void printInfo() const
    {
        int i;
        for (i = 0; i < num; i++)
        {
            cout << vals[i];
        }
        cout << endl;
    }
};
```

EECS  
402

Andrew M Morgan

3



3



## Use Of The DynArrayClass, p.1

EECS  
402

```
int main()
{
    DynArrayClass dal;
    DynArrayClass da2;

    dal.setVals('d', 'r', 'e', 'w');
    cout << "dal: ";
    dal.printInfo();
    da2 = dal;
    cout << "da2 after assigning to dal: ";
    da2.printInfo();
    cout << "Change dal drew => draw" << endl;
    dal.changeVal(2, 'a');
    cout << "dal: ";
    dal.printInfo();
    cout << "da2: ";
    da2.printInfo();

    return 0;
}
```

### Notes:

- Red arrow indicates which instructions have been executed
- Purple filled cells indicate that memory has been allocated by the program for its use

|      |  |          |      |  |
|------|--|----------|------|--|
| 1000 |  | da1.num  | 1018 |  |
| 1001 |  |          | 1019 |  |
| 1002 |  |          | 1020 |  |
| 1003 |  |          | 1021 |  |
| 1004 |  | da1.vals | 1022 |  |
| 1005 |  |          | 1023 |  |
| 1006 |  |          | 1024 |  |
| 1007 |  |          | 1025 |  |
| 1008 |  | da2.num  | 1026 |  |
| 1009 |  |          | 1027 |  |
| 1010 |  |          | 1028 |  |
| 1011 |  |          | 1029 |  |
| 1012 |  | da2.vals | 1030 |  |
| 1013 |  |          | 1031 |  |
| 1014 |  |          | 1032 |  |
| 1015 |  |          | 1033 |  |
| 1016 |  |          | 1034 |  |
| 1017 |  |          | 1035 |  |

EECS  
402

Andrew M Morgan

4



4

M

## Use Of The DynAryClass, p.2

EECS  
402

```

int main()
{
    DynAryClass dal;
    DynAryClass da2;

    dal.setVals('d', 'r', 'e', 'w');
    cout << "dal: ";
    dal.printInfo();
    da2 = dal;
    cout << "da2 after assigning to dal: ";
    da2.printInfo();
    cout << "Change dal drew => draw" << endl;
    dal.changeVal(2, 'a');
    cout << "dal: ";
    dal.printInfo();
    cout << "da2: ";
    da2.printInfo();

    return 0;
}

```

**Notes:**

- Red arrow indicates which instructions have been executed
- Purple filled cells indicate that memory has been allocated by the program for its use

|      |          |      |   |
|------|----------|------|---|
| 1000 | da1.num  | 1018 |   |
| 1001 | 4        | 1019 |   |
| 1002 |          | 1020 |   |
| 1003 |          | 1021 |   |
| 1004 |          | 1022 |   |
| 1005 | da1.vals | 1023 |   |
| 1006 | 1032     | 1024 |   |
| 1007 |          | 1025 |   |
| 1008 |          | 1026 |   |
| 1009 |          | 1027 |   |
| 1010 | da2.num  | 1028 |   |
| 1011 |          | 1029 |   |
| 1012 |          | 1030 |   |
| 1013 |          | 1031 |   |
| 1014 |          | 1032 | d |
| 1015 |          | 1033 | r |
| 1016 |          | 1034 | e |
| 1017 |          | 1035 | w |

EECS  
402

Andrew M Morgan

5 

M

5

M

## Use Of The DynAryClass, p.3

EECS  
402

```

int main()
{
    DynAryClass dal;
    DynAryClass da2;

    dal.setVals('d', 'r', 'e', 'w');
    cout << "dal: ";
    dal.printInfo();
    da2 = dal;
    cout << "da2 after assigning to dal: ";
    da2.printInfo();
    cout << "Change dal drew => draw" << endl;
    dal.changeVal(2, 'a');
    cout << "dal: ";
    dal.printInfo();
    cout << "da2: ";
    da2.printInfo();

    return 0;
}

```

**Notes:**

- Red arrow indicates which instructions have been executed
- Purple filled cells indicate that memory has been allocated by the program for its use

|      |          |      |   |
|------|----------|------|---|
| 1000 | da1.num  | 1018 |   |
| 1001 | 4        | 1019 |   |
| 1002 |          | 1020 |   |
| 1003 |          | 1021 |   |
| 1004 |          | 1022 |   |
| 1005 | da1.vals | 1023 |   |
| 1006 | 1032     | 1024 |   |
| 1007 |          | 1025 |   |
| 1008 |          | 1026 |   |
| 1009 |          | 1027 |   |
| 1010 | da2.num  | 1028 |   |
| 1011 |          | 1029 |   |
| 1012 |          | 1030 |   |
| 1013 |          | 1031 |   |
| 1014 |          | 1032 | d |
| 1015 |          | 1033 | r |
| 1016 |          | 1034 | e |
| 1017 |          | 1035 | w |

EECS  
402

Andrew M Morgan

6 

M

6

M

## Use Of The DynAryClass, p.4

EECS  
402

```

int main()
{
    DynAryClass dal;
    DynAryClass da2;

    dal.setVals('d', 'r', 'e', 'w');
    cout << "dal: ";
    dal.printInfo();
    da2 = dal;
    cout << "da2 after assigning to dal: ";
    da2.printInfo();
    cout << "Change dal drew => draw" << endl;
    dal.changeVal(2, 'a');
    cout << "dal: ";
    dal.printInfo();
    cout << "da2: ";
    da2.printInfo();

    return 0;
}

```

|      |          |      |   |
|------|----------|------|---|
| 1000 | da1.num  | 1018 |   |
| 1001 | 4        | 1019 |   |
| 1002 |          | 1020 |   |
| 1003 |          | 1021 |   |
| 1004 |          | 1022 |   |
| 1005 | da1.vals | 1023 |   |
| 1006 | 1032     | 1024 |   |
| 1007 |          | 1025 |   |
| 1008 |          | 1026 |   |
| 1009 |          | 1027 |   |
| 1010 | da2.num  | 1028 |   |
| 1011 | 4        | 1029 |   |
| 1012 |          | 1030 |   |
| 1013 |          | 1031 |   |
| 1014 |          | 1032 | d |
| 1015 | da2.vals | 1033 | r |
| 1016 | 1032     | 1034 | e |
| 1017 |          | 1035 | w |

Notes:

- Red arrow indicates which instructions have been executed
- Purple filled cells indicate that memory has been allocated by the program for its use

EECS  
402

Andrew M Morgan

7 

M

7

M

## Use Of The DynAryClass, p.5

EECS  
402

```

int main()
{
    DynAryClass dal;
    DynAryClass da2;

    dal.setVals('d', 'r', 'e', 'w');
    cout << "dal: ";
    dal.printInfo();
    da2 = dal;
    cout << "da2 after assigning to dal: ";
    da2.printInfo();
    cout << "Change dal drew => draw" << endl;
    dal.changeVal(2, 'a');
    cout << "dal: ";
    dal.printInfo();
    cout << "da2: ";
    da2.printInfo();

    return 0;
}

```

|      |          |      |   |
|------|----------|------|---|
| 1000 | da1.num  | 1018 |   |
| 1001 | 4        | 1019 |   |
| 1002 |          | 1020 |   |
| 1003 |          | 1021 |   |
| 1004 |          | 1022 |   |
| 1005 | da1.vals | 1023 |   |
| 1006 | 1032     | 1024 |   |
| 1007 |          | 1025 |   |
| 1008 |          | 1026 |   |
| 1009 |          | 1027 |   |
| 1010 | da2.num  | 1028 |   |
| 1011 | 4        | 1029 |   |
| 1012 |          | 1030 |   |
| 1013 |          | 1031 |   |
| 1014 |          | 1032 | d |
| 1015 | da2.vals | 1033 | r |
| 1016 | 1032     | 1034 | e |
| 1017 |          | 1035 | w |

Notes:

- Red arrow indicates which instructions have been executed
- Purple filled cells indicate that memory has been allocated by the program for its use

EECS  
402

Andrew M Morgan

8 

M

8



## Use Of The DynArrayClass, p.6

EECS  
402

```
int main()
{
    DynArrayClass da1;
    DynArrayClass da2;

    da1.setVals('d', 'r', 'e', 'w');
    cout << "da1: ";
    da1.printInfo();
    da2 = da1;
    cout << "da2 after assigning to da1: ";
    da2.printInfo();
    cout << "Change da1 drew => draw" << endl;
    da1.changeVal(2, 'a');
    cout << "da1: ";
    da1.printInfo();
    cout << "da2: ";
    da2.printInfo();

    return 0;
}
```

Notes:

- Red arrow indicates which instructions have been executed
- Purple filled cells indicate that memory has been allocated by the program for its use

da1: drew  
da2 after assigning to da1: drew  
Change da1 drew => draw

| Address | Value | Label    |
|---------|-------|----------|
| 1000    |       | da1.num  |
| 1001    | 4     |          |
| 1002    |       |          |
| 1003    |       |          |
| 1004    |       | da1.vals |
| 1005    | 1032  |          |
| 1006    |       |          |
| 1007    |       |          |
| 1008    |       | da2.num  |
| 1009    | 4     |          |
| 1010    |       |          |
| 1011    |       |          |
| 1012    |       | da2.vals |
| 1013    | 1032  |          |
| 1014    |       |          |
| 1015    |       |          |
| 1016    |       |          |
| 1017    |       |          |
| 1018    |       |          |
| 1019    |       |          |
| 1020    |       |          |
| 1021    |       |          |
| 1022    |       |          |
| 1023    |       |          |
| 1024    |       |          |
| 1025    |       |          |
| 1026    |       |          |
| 1027    |       |          |
| 1028    |       |          |
| 1029    |       |          |
| 1030    |       |          |
| 1031    |       |          |
| 1032    | d     |          |
| 1033    | r     |          |
| 1034    | a     |          |
| 1035    | w     |          |

EECS  
402

Andrew M Morgan

9



9



## Use Of The DynArrayClass, p.7

EECS  
402

```
int main()
{
    DynArrayClass da1;
    DynArrayClass da2;

    da1.setVals('d', 'r', 'e', 'w');
    cout << "da1: ";
    da1.printInfo();
    da2 = da1;
    cout << "da2 after assigning to da1: ";
    da2.printInfo();
    cout << "Change da1 drew => draw" << endl;
    da1.changeVal(2, 'a');
    cout << "da1: ";
    da1.printInfo();
    cout << "da2: ";
    da2.printInfo();

    return 0;
}
```

da1: drew  
da2 after assigning to da1: drew  
Change da1 drew => draw  
da1: draw  
da2: draw

da2 seems to have changed, but code only changed da1!!

| Address | Value | Label    |
|---------|-------|----------|
| 1000    |       | da1.num  |
| 1001    | 4     |          |
| 1002    |       |          |
| 1003    |       |          |
| 1004    |       | da1.vals |
| 1005    | 1032  |          |
| 1006    |       |          |
| 1007    |       |          |
| 1008    |       | da2.num  |
| 1009    | 4     |          |
| 1010    |       |          |
| 1011    |       |          |
| 1012    |       | da2.vals |
| 1013    | 1032  |          |
| 1014    |       |          |
| 1015    |       |          |
| 1016    |       |          |
| 1017    |       |          |
| 1018    |       |          |
| 1019    |       |          |
| 1020    |       |          |
| 1021    |       |          |
| 1022    |       |          |
| 1023    |       |          |
| 1024    |       |          |
| 1025    |       |          |
| 1026    |       |          |
| 1027    |       |          |
| 1028    |       |          |
| 1029    |       |          |
| 1030    |       |          |
| 1031    |       |          |
| 1032    | d     |          |
| 1033    | r     |          |
| 1034    | a     |          |
| 1035    | w     |          |

EECS  
402

Andrew M Morgan

10



10

- The member-by-member assignment operator copied only the pointer value
  - This results in multiple objects pointing to the same physical memory location
  - When one object changes the memory, change is seen on other objects since that memory is "shared" between the objects
- Programmer must **override the default assignment** behavior to prevent this, if it is a problem!
  - Provide your own implementation of the assignment operator that does more than a member-by-member copy
  - Create a whole new array for the left-hand-side object and copy the contents over
- Most often, when you overload the assignment operator in this way, you will want to overload the copy ctor too
  - Default copy ctor has same problem since it does a member-by-member copy as well

```

class DynArrayClass
{
private:
    int num;
    char *vals; //Will pt to array of chars
public:
    ///---No changes to other functions---
    //Overloaded assignment operator
    void operator=(const DynArrayClass &rhs)
    {
        int i;
        num = rhs.num; //Just copy the static vars
        vals = new char[num];
        for (i = 0; i < num; i++)
        {
            vals[i] = rhs.vals[i]; //Copy ary contents
        }
    }
    //Copy ctor implementation
    DynArrayClass(const DynArrayClass &rhs)
    {
        int i;
        num = rhs.num; //Just copy the static vars
        vals = new char[num];
        for (i = 0; i < num; i++)
        {
            vals[i] = rhs.vals[i]; //Copy ary contents
        }
    }
    DynArrayClass():num(0), vals(0)
    { ; }
};

```

Remember – when you provide a constructor, the "implicit default ctor" is no longer available! Therefore, when adding the copy ctor, another ctor must be overloaded as well to allow construction of new objects.

NOTE: This should have been done earlier anyhow, since pointers should be initialized to 0 (NULL) and this didn't happen in the earlier versions of the program in this lecture!



## Using The New DynAryClass, p.1

EECS  
402

```
int main()
{
    DynAryClass da1;
    DynAryClass da2;

    da1.setVals('d', 'r', 'e', 'w');
    cout << "da1: ";
    da1.printInfo();
    da2 = da1;
    cout << "da2 after assigning to da1: ";
    da2.printInfo();
    cout << "Change da1 drew => draw" << endl;
    da1.changeVal(2, 'a');
    cout << "da1: ";
    da1.printInfo();
    cout << "da2: ";
    da2.printInfo();

    return 0;
}
```

### Notes:

- Red arrow indicates which instructions have been executed
- Purple filled cells indicate that memory has been allocated by the program for its use

da1: drew

|      |          |      |   |
|------|----------|------|---|
| 1000 | da1.num  | 1018 |   |
| 1001 | 4        | 1019 |   |
| 1002 |          | 1020 |   |
| 1003 |          | 1021 |   |
| 1004 | da1.vals | 1022 |   |
| 1005 | 1032     | 1023 |   |
| 1006 |          | 1024 |   |
| 1007 |          | 1025 |   |
| 1008 | da2.num  | 1026 |   |
| 1009 |          | 1027 |   |
| 1010 |          | 1028 |   |
| 1011 |          | 1029 |   |
| 1012 | da2.vals | 1030 |   |
| 1013 |          | 1031 |   |
| 1014 |          | 1032 | d |
| 1015 |          | 1033 | r |
| 1016 |          | 1034 | e |
| 1017 |          | 1035 | w |

EECS  
402

Andrew M Morgan

13



13



## Using The New DynAryClass, p.2

EECS  
402

```
int main()
{
    DynAryClass da1;
    DynAryClass da2;

    da1.setVals('d', 'r', 'e', 'w');
    cout << "da1: ";
    da1.printInfo();
    da2 = da1;
    cout << "da2 after assigning to da1: ";
    da2.printInfo();
    cout << "Change da1 drew => draw" << endl;
    da1.changeVal(2, 'a');
    cout << "da1: ";
    da1.printInfo();
    cout << "da2: ";
    da2.printInfo();

    return 0;
}
```

### Notes:

- Red arrow indicates which instructions have been executed
- Purple filled cells indicate that memory has been allocated by the program for its use

da1: drew

|      |          |      |   |
|------|----------|------|---|
| 1000 | da1.num  | 1018 |   |
| 1001 | 4        | 1019 |   |
| 1002 |          | 1020 |   |
| 1003 |          | 1021 |   |
| 1004 | da1.vals | 1022 |   |
| 1005 | 1032     | 1023 |   |
| 1006 |          | 1024 |   |
| 1007 |          | 1025 |   |
| 1008 | da2.num  | 1026 |   |
| 1009 | 4        | 1027 |   |
| 1010 |          | 1028 | d |
| 1011 |          | 1029 | r |
| 1012 |          | 1030 | e |
| 1013 | 1028     | 1031 | w |
| 1014 |          | 1032 | d |
| 1015 |          | 1033 | r |
| 1016 |          | 1034 | e |
| 1017 |          | 1035 | w |

EECS  
402

Andrew M Morgan

14



14



## Using The New DynAryClass, p.3

EECS  
402

```

int main()
{
    DynAryClass da1;
    DynAryClass da2;

    da1.setVals('d', 'r', 'e', 'w');
    cout << "da1: ";
    da1.printInfo();
    da2 = da1;
    cout << "da2 after assigning to da1: ";
    → da2.printInfo();
    cout << "Change da1 drew => draw" << endl;
    da1.changeVal(2, 'a');
    cout << "da1: ";
    da1.printInfo();
    cout << "da2: ";
    da2.printInfo();

    return 0;
}

```

Notes:

- Red arrow indicates which instructions have been executed
- Purple filled cells indicate that memory has been allocated by the program for its use

|      |          |      |   |
|------|----------|------|---|
| 1000 | da1.num  | 1018 |   |
| 1001 | 4        | 1019 |   |
| 1002 |          | 1020 |   |
| 1003 |          | 1021 |   |
| 1004 | da1.vals | 1022 |   |
| 1005 | 1032     | 1023 |   |
| 1006 |          | 1024 |   |
| 1007 |          | 1025 |   |
| 1008 | da2.num  | 1026 |   |
| 1009 | 4        | 1027 |   |
| 1010 |          | 1028 | d |
| 1011 |          | 1029 | r |
| 1012 | da2.vals | 1030 | e |
| 1013 | 1028     | 1031 | w |
| 1014 |          | 1032 | d |
| 1015 |          | 1033 | r |
| 1016 |          | 1034 | e |
| 1017 |          | 1035 | w |

da1: drew  
da2 after assigning to da1: drew

EECS  
402

Andrew M Morgan

15



15



## Using The New DynAryClass, p.4

EECS  
402

```

int main()
{
    DynAryClass da1;
    DynAryClass da2;

    da1.setVals('d', 'r', 'e', 'w');
    cout << "da1: ";
    da1.printInfo();
    da2 = da1;
    cout << "da2 after assigning to da1: ";
    da2.printInfo();
    cout << "Change da1 drew => draw" << endl;
    da1.changeVal(2, 'a');
    → cout << "da1: ";
    da1.printInfo();
    cout << "da2: ";
    da2.printInfo();

    return 0;
}

```

Notes:

- Red arrow indicates which instructions have been executed
- Purple filled cells indicate that memory has been allocated by the program for its use

|      |          |      |   |
|------|----------|------|---|
| 1000 | da1.num  | 1018 |   |
| 1001 | 4        | 1019 |   |
| 1002 |          | 1020 |   |
| 1003 |          | 1021 |   |
| 1004 | da1.vals | 1022 |   |
| 1005 | 1032     | 1023 |   |
| 1006 |          | 1024 |   |
| 1007 |          | 1025 |   |
| 1008 | da2.num  | 1026 |   |
| 1009 | 4        | 1027 |   |
| 1010 |          | 1028 | d |
| 1011 |          | 1029 | r |
| 1012 | da2.vals | 1030 | e |
| 1013 | 1028     | 1031 | w |
| 1014 |          | 1032 | d |
| 1015 |          | 1033 | r |
| 1016 |          | 1034 | a |
| 1017 |          | 1035 | w |

da1: drew  
da2 after assigning to da1: drew  
Change da1 drew => draw

EECS  
402

Andrew M Morgan

16



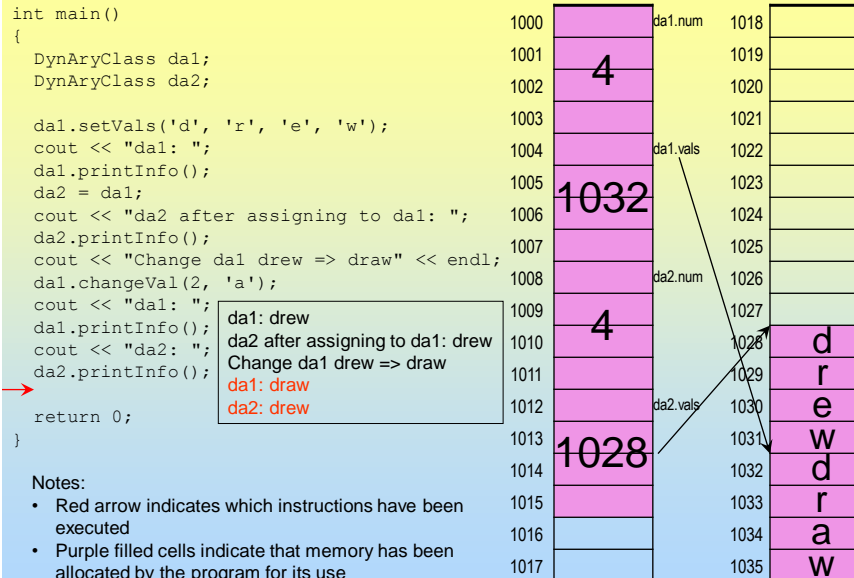
16





## Using The New DynAryClass, p.5

EECS  
402



EECS  
402

Andrew M Morgan

17



17



## General Rule, Version 1

EECS  
402

- When a class you write contains data members that will point to dynamically allocated memory:
  - Override the assignment operator so that a full copy of the object (and all dynamic memory associated with it) is made when assigning
    - This will take the place of (override) the default assignment operator that performs a shallow copy
  - Override the copy constructor so that a full copy of the object (and all dynamic memory associated with it) is made when copying
    - This will take the place of (override) the default copy ctor that performs a shallow copy
  - Overload another constructor (default or otherwise) so that objects can be constructed with something other than the copy ctor
  - Initialize the pointer members to 0 (NULL) in the ctors

EECS  
402

Andrew M Morgan

18



18



## Yet Another Problem, p.1

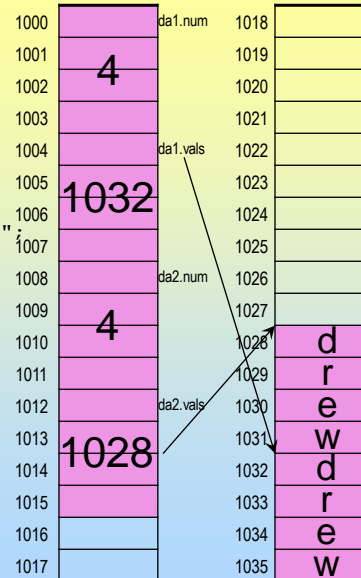
EECS  
402

```
int main()
{
    DynAryClass da1;
    DynAryClass da2;

    da1.setVals('d', 'r', 'e', 'w');
    cout << "da1: ";
    da1.printInfo();
    da2 = da1;
    cout << "da2 after assigning to da1: ";
    da2.printInfo();
    da1.setVals('h', 'i', '!');
    cout << "newly set da1: ";
    da1.printInfo();
    da2.setVals('a', 'b', 'c');
    cout << "newly set da2: ";
    da2.printInfo();

    return 0;
}
```

da1: drew  
da2 after assigning to da1: drew



EECS  
402

Andrew M Morgan

19



19



## Yet Another Problem, p.2

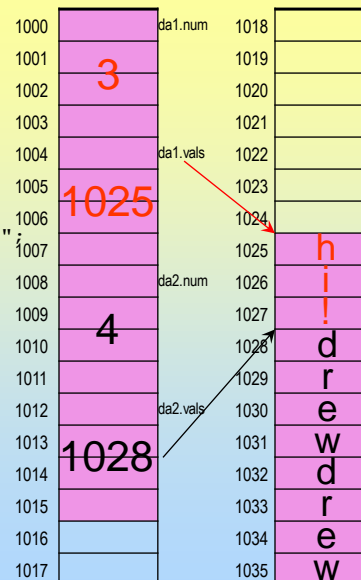
EECS  
402

```
int main()
{
    DynAryClass da1;
    DynAryClass da2;

    da1.setVals('d', 'r', 'e', 'w');
    cout << "da1: ";
    da1.printInfo();
    da2 = da1;
    cout << "da2 after assigning to da1: ";
    da2.printInfo();
    da1.setVals('h', 'i', '!');
    cout << "newly set da1: ";
    da1.printInfo();
    da2.setVals('a', 'b', 'c');
    cout << "newly set da2: ";
    da2.printInfo();

    return 0;
}
```

da1: drew  
da2 after assigning to da1: drew  
newly set da1: hi!



EECS  
402

Andrew M Morgan

20



20



## Yet Another Problem, p.3

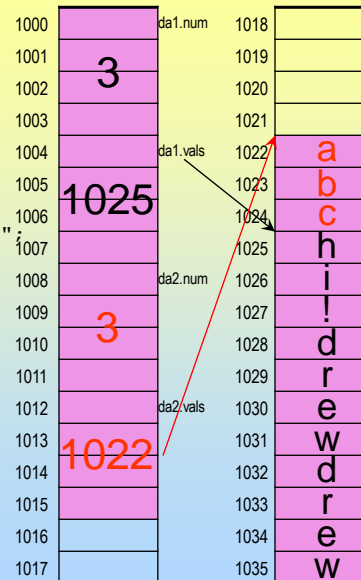
EECS  
402

```
int main()
{
    DynAryClass da1;
    DynAryClass da2;

    da1.setVals('d', 'r', 'e', 'w');
    cout << "da1: ";
    da1.printInfo();
    da2 = da1;
    cout << "da2 after assigning to da1: ";
    da2.printInfo();
    da1.setVals('h', 'i', '!');
    cout << "newly set da1: ";
    da1.printInfo();
    da2.setVals('a', 'b', 'c');
    cout << "newly set da2: ";
    da2.printInfo();

    return 0;
}
```

da1: drew  
da2 after assigning to da1: drew  
newly set da1: hi!  
newly set da2: abc



EECS  
402

Andrew M Morgan

21



21



## Yet Another Problem, p.4

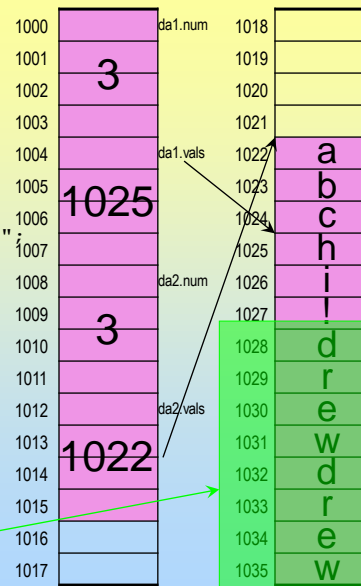
EECS  
402

```
int main()
{
    DynAryClass da1;
    DynAryClass da2;

    da1.setVals('d', 'r', 'e', 'w');
    cout << "da1: ";
    da1.printInfo();
    da2 = da1;
    cout << "da2 after assigning to da1: ";
    da2.printInfo();
    da1.setVals('h', 'i', '!');
    cout << "newly set da1: ";
    da1.printInfo();
    da2.setVals('a', 'b', 'c');
    cout << "newly set da2: ";
    da2.printInfo();

    return 0;
}
```

This memory has been "leaked".  
There is no way to access it!!! The pointers  
that used to point to it have changed.



EECS  
402

Andrew M Morgan

22



22



## Plugging The Memory Leak

EECS  
402

- You must be sure to delete memory when you will no longer use it!
- Use the delete operator to free up memory before re-allocating new memory

```
void setVals(char a, char b,
             char c)
{
    num = 3;
    delete [] vals;
    vals = new char[num];
    vals[0] = a;
    vals[1] = b;
    vals[2] = c;
}

void setVals(char a, char b,
             char c, char d)
{
    num = 4;
    delete [] vals;
    vals = new char[num];
    vals[0] = a;
    vals[1] = b;
    vals[2] = c;
    vals[3] = d;
}
```

Recall that using delete on a null pointer will have no effect (and will not seg fault). Using delete on a pointer that points to memory not owned by your program WILL cause a seg fault.

Therefore, it is important that the "vals" member be initialized to NULL so that the first time these functions are used, vals is a null pointer and the delete operator will not cause a seg fault.

EECS  
402

Andrew M Morgan

23



23



## One Final Problem, p.1

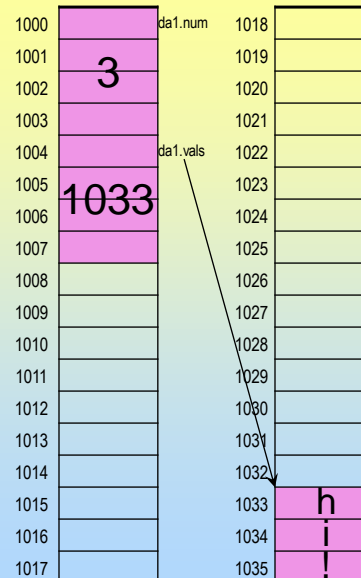
EECS  
402

```
//Notes: Not a member function!
//      da is passed-by-value
void globalPrintDynAry(DynAryClass da)
{
    cout << "Global Print: ";
    da.printInfo();
}

int main()
{
    DynAryClass da1;
    DynAryClass da2;

    da1.setVals('h', 'i', '!');
    cout << "newly set da1: ";
    da1.printInfo();
    → globalPrintDynAry(da1);

    return 0;
}
```



EECS  
402

Andrew M Morgan

24



24



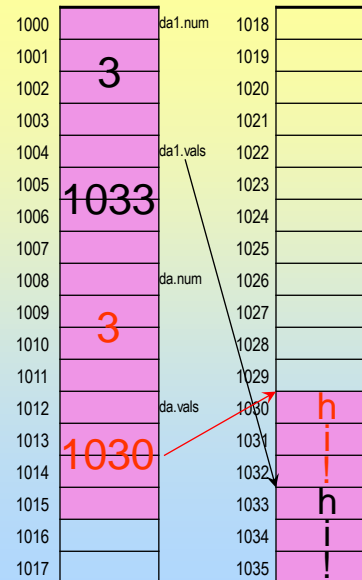
## One Final Problem, p.2

EECS  
402

```
//Notes: Not a member function!  
//      da is passed-by-value  
void globalPrintDynAry(DynAryClass da)  
{  
    cout << "Global Print: ";  
    da.printInfo();  
}  
  
int main()  
{  
    DynAryClass dal;  
    DynAryClass da2;  
  
    dal.setVals('h', 'i', '!');  
    cout << "newly set dal: ";  
    dal.printInfo();  
    globalPrintDynAry(dal);  
  
    return 0;  
}
```

newly set da1: hi!

Pass-by-value param uses the  
(programmer-defined) copy ctor  
to make a full copy



EECS  
402

Andrew M Morgan

25



25

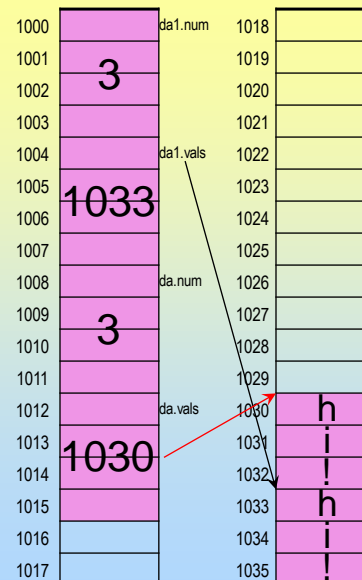


## One Final Problem, p.3

EECS  
402

```
//Notes: Not a member function!  
//      da is passed-by-value  
void globalPrintDynAry(DynAryClass da)  
{  
    cout << "Global Print: ";  
    da.printInfo();  
}  
  
int main()  
{  
    DynAryClass dal;  
    DynAryClass da2;  
  
    dal.setVals('h', 'i', '!');  
    cout << "newly set dal: ";  
    dal.printInfo();  
    globalPrintDynAry(dal);  
  
    return 0;  
}
```

newly set da1: hi!  
Global Print: hi!



EECS  
402

Andrew M Morgan

26



26



## One Final Problem, p.4

EECS  
402

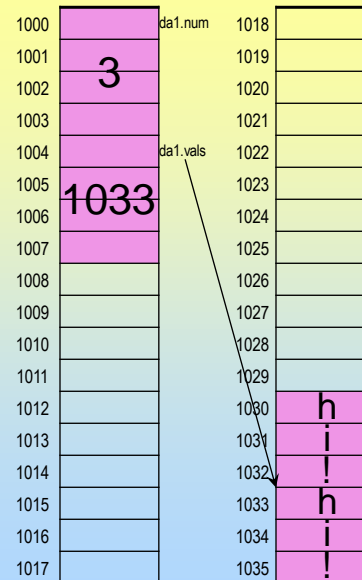
```
//Notes: Not a member function!
//      da is passed-by-value
void globalPrintDynAry(DynAryClass da)
{
    cout << "Global Print: ";
    da.printInfo();
}

int main()
{
    DynAryClass da1;
    DynAryClass da2;

    da1.setVals('h', 'i', '!');
    cout << "newly set da1: hi!";
    da1.printInfo();
    globalPrintDynAry(da1);
    return 0;
}
```

newly set da1: hi!  
Global Print: hi!

The value param "da" goes out of scope, and is therefore removed from memory. The dynamic memory it was pointing to remains though! This is another memory leak.



EECS  
402

Andrew M Morgan

27



27



## Plugging Memory Leak #2

EECS  
402

- The memory leak was caused by an object being destroyed, but the dynamic memory associated with it remaining allocated
- Every time an object is destroyed, the destructor is called
- Solution: Provide a destructor that frees up dynamic memory associated with an object

```
class DynAryClass
{
private:
    int num;
    char *vals; //Will pt to array of chars
public:
    //---No changes to other functions---

    //Dtor - frees up dynamic memory associated with
    //the object being destroyed
    ~DynAryClass()
    {
        delete [] vals;
    }
};
```

EECS  
402

Andrew M Morgan

28



28

- When a class you write contains data members that will point to dynamically allocated memory:
  - Override the assignment operator so that a full copy of the object (and all dynamic memory associated with it) is made when assigning
    - This will take the place of (override) the default assignment operator that performs a shallow copy
  - Override the copy constructor so that a full copy of the object (and all dynamic memory associated with it) is made when copying
    - This will take the place of (override) the default copy ctor that performs a shallow copy
  - Overload another constructor (default or otherwise) so that objects can be created with something other than the copy ctor
  - Initialize the pointer members to 0 (NULL) in the ctors
  - Provide a destructor so that dynamic memory used by an object is freed when the object is destroyed