Problem 1

```
int foo = 0;
int *ptr = &foo;
```

Q: Which of the following statements change foo to 1?

(a) ptr++;
(b) foo++;
(c) (*foo)++;
(d) (*ptr)++;
(e) (a) and (b)
(f) (a) and (d)
(g) (b) and (d)
(h) (c) and (d)

Problem 1

```
int foo = 0;
int *ptr = &foo;
```

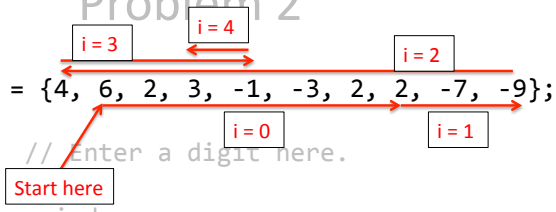
Q: Which of the following statements change foo to 1?

(a) ptr++;
(b) **foo++;**
(c) (*foo)++;
(d) **(*ptr)++;**
(e) (a) and (b)
(f) (a) and (d)
(g) (b) and (d)
(h) (c) and (d)

Problem 2

```
int array[10] = {4, 6, 2, 3, -1, -3, 2, 2, -7, -9};
int index;
cin >> index;  // Enter a digit here.

int *p = array + index;

for (int i = 0; i < 5; i++)
{
  int hops = *p;
  p += hops;
}

cout << *p << endl;
```

Problem 2




```
int array[10] = {4, 6, 2, 3, -1, -3, 2, 2, -7, -9};
int index;
cin >> index;  // Enter a digit here.

int *p = array + index;

for (int i = 0; i < 5; i++)
{
  int hops = *p;
  p += hops;
}

cout << *p << endl;
```

Problem 3

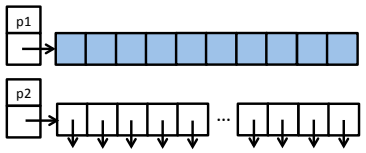
```
int *p1 = new int[10];
int *p2[15];

for (int i = 0; i < 15; i++)
    p2[i] = new int[5];

int **p3 = new int*[5];

for (int i = 0; i < 5; i++)
    p3[i] = new int;

int *p4 = new int;
int *temp = p4;
p4 = p1;
p1 = temp;
```

Problem 3

```
int *p1 = new int[10];
int *p2[15];

for (int i = 0; i < 15; i++)
    p2[i] = new int[5];

int **p3 = new int*[5];

for (int i = 0; i < 5; i++)
    p3[i] = new int;

int *p4 = new int;
int *temp = p4;
p4 = p1;
p1 = temp;
```

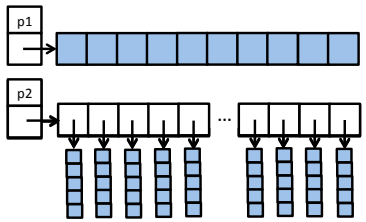
Problem 3

```
int *p1 = new int[10];
int *p2[15];

for (int i = 0; i < 15; i++)
    p2[i] = new int[5];

int **p3 = new int*[5];

for (int i = 0; i < 5; i++)
    p3[i] = new int;

int *p4 = new int;
int *temp = p4;
p4 = p1;
p1 = temp;
```



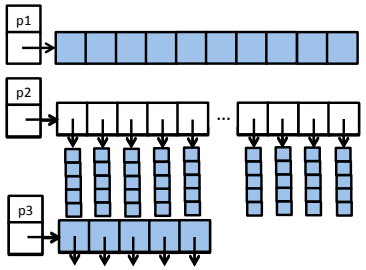
Problem 3

```
int *p1 = new int[10];
int *p2[15];

for (int i = 0; i < 15; i++)
    p2[i] = new int[5];

int **p3 = new int*[5];

for (int i = 0; i < 5; i++)
    p3[i] = new int;

int *p4 = new int;
int *temp = p4;
p4 = p1;
p1 = temp;
```



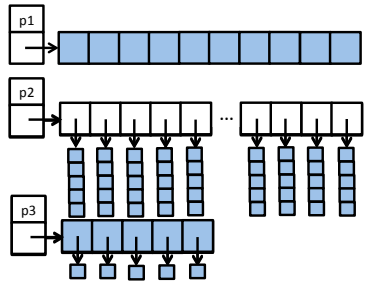
Problem 3

```
int *p1 = new int[10];
int *p2[15];

for (int i = 0; i < 15; i++)
    p2[i] = new int[5];

int **p3 = new int*[5];

for (int i = 0; i < 5; i++)
    p3[i] = new int;

int *p4 = new int;
int *temp = p4;
p4 = p1;
p1 = temp;
```



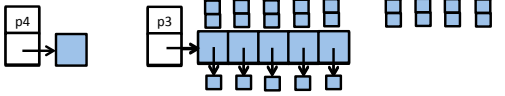
Problem 3

```
int *p1 = new int[10];
int *p2[15];

for (int i = 0; i < 15; i++)
    p2[i] = new int[5];

int **p3 = new int*[5];

for (int i = 0; i < 5; i++)
    p3[i] = new int;

int *p4 = new int;
int *temp = p4;
p4 = p1;
p1 = temp;
```



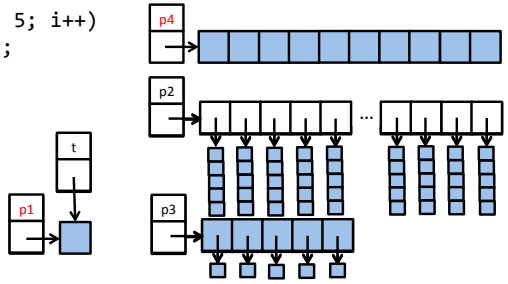
Problem 3

```
int *p1 = new int[10];
int *p2[15];

for (int i = 0; i < 15; i++)
    p2[i] = new int[5];

int **p3 = new int*[5];

for (int i = 0; i < 5; i++)
    p3[i] = new int;

int *p4 = new int;
int *temp = p4;
p4 = p1;
p1 = temp;
```



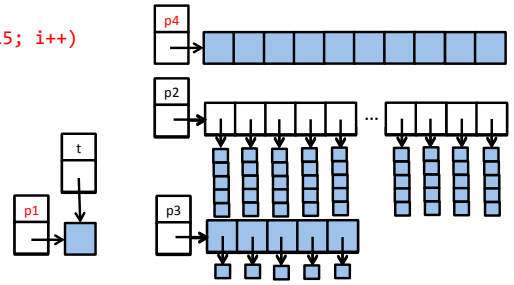
Problem 3

```
int *p1 = new int[10];
int *p2[15];

for (int i = 0; i < 15; i++)
    p2[i] = new int[5];

int **p3 = new int*[5];

for (int i = 0; i < 5; i++)
    p3[i] = new int;

int *p4 = new int;
int *temp = p4;
p4 = p1;
p1 = temp;
```



Problem 3

```
delete p1; // or "delete temp;”

for (int i = 0; i < 5; i++)
    delete p3[i];

delete[] p3;  // This must happen AFTER
              // the above for loop.

for (int i = 0; i < 15; i++)
    delete[] p2[i];

delete[] p4;
```



Problem 4

```
void countMatches(const char *str1, const char *str2, int &count)
{
    *count = 0;

    while (str1 != '\0' || str2 != '\0')
    {
        if (*str1 == *str2)
            *count++;
        str1++;
        str2++;
    }
}
```

Try to understand the intention of the code.
Do not rewrite the whole thing.

Problem 4

```
void countMatches(const char *str1, const char *str2, int &count)
{
    *count = 0;  // Reset count to 0.

    while (str1 != '\0' || str2 != '\0') // Neither string ended.
    {
        if (*str1 == *str2)    // Compare the chars.
            *count++;          // If they are the same, increment count.
        str1++;                // Advance str1.
        str2++;                // Advance str2.
    }
}
```

Try to understand the intention of the code:
Move str1 and str2 in parallel and compare the characters one by one, counting if they are the same.

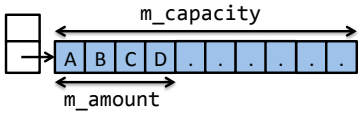
Problem 4

```
void countMatches(const char *str1, const char *str2, int &count)
{
    count = 0;  // Reset count to 0.

    while (*str1 != '\0' && *str2 != '\0')  // Neither string ended.
    {
        if (*str1 == *str2)    // Compare the chars.
            count++;           // If they are the same, increment count.
        str1++;                // Advance str1.
        str2++;                // Advance str2.
    }
}
```

‘&’ here means “reference,” not “address”

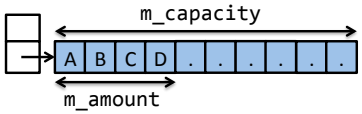
Problem 5

```
class Goldfish
{
  public:
    Goldfish(int capacity);
    ~Goldfish();
    void remember(char c);
    void forget();            // Clears m_memory using dots('.')
    void printMemory() const; // Prints the content of m_memory
  private:
    char *m_memory; // Pointer to memory.
    int m_amount;   // # of chars remembered.
    int m_capacity; // # of chars this fish can remember.
};
```



Problem 5(a)

```
Goldfish::Goldfish(int capacity)
{
    if (capacity < 1)
        m_capacity = 3;
    else
        m_capacity = capacity;

    m_memory = new char[m_capacity];
    m_amount = 0;
    forget();
};
```

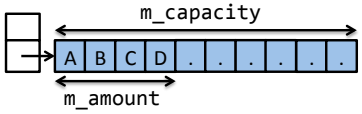


Problem 5(a)

```
Goldfish::Goldfish(int capacity)
{
    if (capacity < 1)
        m_capacity = 3;
    else
        m_capacity = capacity;

    m_memory = new char[m_capacity];
    m_amount = 0;
    forget();
};
```

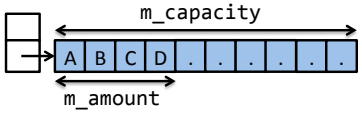
After writing the definition, ask yourself:

Have I initialized everything that I must initialize? (Look at the private member variables and see if all of them are initialized properly.)



Problem 5(b)

```
void Goldfish::remember(char c)
{
    if (m_amount == m_capacity)
    {
       for (int i = 0; i < m_capacity - 1; i++)
            m_memory[i] = m_memory[i + 1];
        m_amount--;
    }
    m_memory[m_amount] = c;
    m_amount++;
}
```



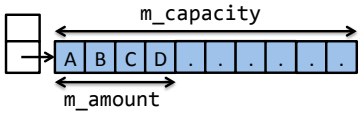
Problem 5(b)

```
void Goldfish::remember(char c)
{
    if (m_amount == m_capacity)
    {
       for (int i = 0; i < m_capacity - 1; i
            m_memory[i] = m_memory[i + 1];
        m_amount--;
    }
    m_memory[m_amount] = c;
    m_amount++;
}
```

After writing the definition, ask yourself:

Have I updated everything that must be updated?

m_capacity: shouldn't change.
m_memory: shouldn't change.
m_amount: incremented if a new character is added.



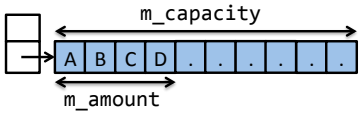
Problem 5(c)

```
void Goldfish::forget()
{
    for (int i = 0; i < m_capacity; i++)
        m_memory[i] = '.';
    m_amount = 0;
 }
```

After writing the definition, ask yourself:

Have I updated everything that must be updated?

m_capacity: shouldn't change.
m_memory: shouldn't change.
m_amount: set to 0.

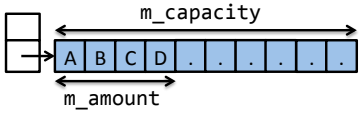


Problem 5(d)

```
Goldfish::~Goldfish()
{
  delete[] m_memory;
}
```

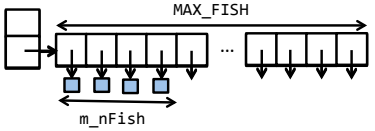
After writing the definition, ask yourself:

Have I deleted all dynamically allocated objects that are created in this class?



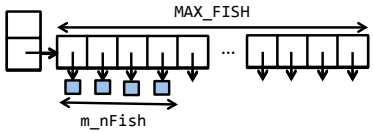
Problem 6

```
const int MAX_FISH = 20;

class Aquarium
{
  public:
    Aquarium();
    bool addFish(int capacity);
    Goldfish *getFish(int n);
    void oracle();
    ~Aquarium();
  private:
    Goldfish *m_fish[MAX_FISH];  // Pointers to fish.
    int m_nFish;                 // Number of Fish.
};
```



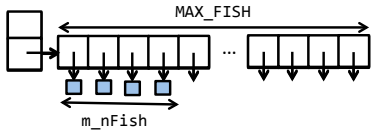
Problem 6(a)

```
Aquarium::Aquarium()
{
    m_nFish = 0;
}

OR

Aquarium::Aquarium()
: m_nFish(0)
{}
```



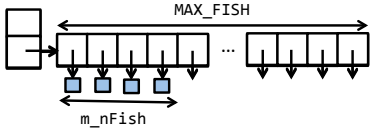
Problem 6(b)

```
bool Aquarium::addFish(int capacity)
{
    if (m_nFish >= MAX_FISH)
        return false;

    m_fish[m_nFish] = new Goldfish(capacity);
    m_nFish++;
    return true;
}
```



Problem 6(c)

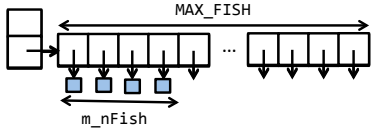
```
Goldfish* Aquarium::getFish(int n)
{
    if (n < 0 || n >= m_nFish)
        return NULL;

    return m_fish[n];
}
```



Problem 6(d)

```
Aquarium::~Aquarium()
{
    for (int i = 0; i < m_nFish; i++)
        delete m_fish[i];
}
```

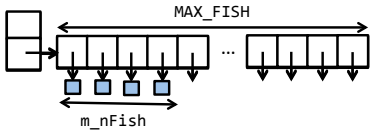
Why not "`delete[] m_fish;`"?



Problem 6(d)

```
void Aquarium::oracle()
{
  for (int i = 0; i < m_nFish; i++)
  {
    m_fish[i]->printMemory();
    m_fish[i]->forget();
  }
}
```

Why not "`m_fish[i].printMemory();`"?



Problem 7

1. Name of the class: `BankAccount`
2. No default constructor: No `BankAccount()`.
3. One constructor that takes in initial balance and the password. A password is an integer:
   `BankAccount(double initAmt, int pwd);`
   The `BankAccount` must keep track of the balance and the password.
   `double m_balance;`
   `int m_password;`
4. `There are` member functions `deposit` and `withdraw`. Each function takes an amount of money and the password. If the password is wrong, `deposit/withdraw` returns `false`.
   `bool deposit(double amt, int pwd);`
   `bool withdraw(double amt, int pwd);`
5. `setPassword` takes in two passwords.
   `bool setPassword(int oldPwd, int newPwd);`
6. `balance` returns the current balance if the password is correct, -1 otherwise.
   `double balance(int pwd) const;`

Problem 7

1. Name of the class: `BankAccount`
2. No default constructor: No `BankAccount()`.
3. One constructor that takes in initial balance and the password. A password is an integer:
   `BankAccount(double initAmt, int pwd);`
   The `BankAccount` must keep track of the balance and the password.
   `double m_balance;`
   `int m_password;`
4. `There are` member functions `deposit` and `withdraw`. Each function takes an amount of money and the password. If the password is wrong, `deposit/withdraw` returns `false`.
   `bool deposit(double amt, int pwd);`
   `bool withdraw(double amt, int pwd);`
5. `setPassword` takes in two passwords.
   `bool setPassword(int oldPwd, int newPwd);`
6. `balance` returns the current balance if the password is correct, -1 otherwise.
   `double balance(int pwd) const;`

why const?

Problem 7

```
class BankAccount
{
  public:
    BankAccount(double initAmt, int pwd);
    double balance(int pwd) const;
    bool deposit(double amt, int pwd);
    bool withdraw(double amt, int pwd);
    bool setPassword(int oldPwd, int newPwd);
  private:
    double m_balance;
    int m_password;
};
```

I will skip the rest, since they're pretty easy.

Problem 8 and 9

- More class exercises.

Bonus Problem

- (A simplified version of) the Wheel of Fortune game.
- Skeleton code is posted, along with the solution.
- Involves C strings, dynamic allocation, and class.