# CS 31 Introduction to CS Discussion 2H

# Practice

---

Problem #1: There is a Vending Machine Class called VM below. Please complete the codes under the comment TODO. The output of this program is:

**I bought Coke**

**Pepsi is sold out!!**

---

```
#include <iostream>
#include <string>
using namespace std;
const int MAXSODA = 100;

class Soda
{
public:
        Soda();
        void setName(string name);
        string getName() const;
private:
        string name;
};

Soda::Soda()
{
        // TODO: set name = "NA" to signal Not Available (5 points)


}
void Soda::setName(string name)
{
        // TODO: set Soda's name as the name passed in. (5 points)


}
```

```
string Soda::getName() const
{
        // TODO: return Soda's name. (5 points)


}
```

```
Soda::Soda()
{
        // TODO: set name = "NA" to signal Not Available (5 points)
        this->name = "NA";
}
void Soda::setName(string name)
{
        // TODO: set Soda's name as the name passed in. (5 points)
        this->name = name;
        // We have to use this->name here.
}
string Soda::getName() const
{
        // TODO: return Soda's name. (5 points)
        return name;
        // or return this->name;
}
```

```
class VM
{
public:
        VM(int n);
        ~VM();
        void restock(string name,int quantity);
        Soda* getSoda(string name);
        bool buySoda(string name);
private:
        Soda* inventory[MAXSODA];
        int quantity[MAXSODA];
        int numSoda;
};

VM::VM(int n)
{
        numSoda = n;

        for(int i=0;i<numSoda;i++)
                inventory[ i ] = new Soda();
}

VM::~VM()
{
        // TODO: (10 points)
        // Delete the storage pointed to by Soda pointers in inventory array




}
```

```
class VM
{
public:
        VM(int n);
        ~VM();
        void restock(string name,int quantity);
        Soda* getSoda(string name);
        bool buySoda(string name);
private:
        Soda* inventory[MAXSODA];
        int quantity[MAXSODA];
        int numSoda;
};

VM::VM(int n)
{
        numSoda = n;

        for(int i=0;i<numSoda;i++)
                inventory[ i ] = new Soda();
}

VM::~VM()
{
        // TODO: (10 points)
        // Delete the storage pointed to by Soda pointers in inventory array

        for(int i=0;i<numSoda;i++)
                delete inventory[i];

}
```

```
void VM::restock(string name,int quantity)
{
        for(int i=0;i<numSoda;i++)
                if( inventory[i]->getName() == "NA") {
                        // TODO: (10 points)
                        // 1. If we found a Soda that has the name "NA", we set
                        // this Soda to have the name we passed into this function.
                        // 2. Set the quantity for that Soda.
                        // 3. Break out of this loop.




                }
}

Soda* VM::getSoda(string name)
{

        // TODO: (10 points)
        // Search all Soda Objects to see if there's a matching Soda by name
        //    If there is a matching soda by name, return that Soda object.
        //    Return nullptr if there's no matching Soda by name.









}
```

```
void VM::restock(string name,int quantity)
{
        for(int i=0;i<numSoda;i++)
                if( inventory[i]->getName() == "NA") {
                        // TODO: (10 points)
                        // 1. If we found a Soda that has the name "NA", we set
                        // this Soda to have the name we passed into this function.
                        // 2. Set the quantity for that Soda.
                        // 3. Break out of this loop.
                        inventory[i]->setName(name);

                        this->quantity[i] = quantity;

                        break;   // or return;
                }
}

Soda* VM::getSoda(string name)
{

        // TODO: (10 points)
        // Search all Soda Objects to see if there's a matching Soda by name
        //    If there is a matching soda by name, return that Soda object.
        //    Return nullptr if there's no matching Soda by name.

        for(int i=0;i<numSoda;i++)
                if( inventory[i]->getName() == name)
                        return inventory[i];

        return nullptr;


}
```

```
bool VM::buySoda(string name)
{
        // TODO: (15 points)
        // 1. Search through all Soda objects to find matching Soda by name
        //    If there is a matching soda by name, and if the quantity is > 0.
        //    then we decrease quantity for the Soda by 1 and return true
        //    If there is a matching soda by name, but the quantity is <= 0
        //    then we return false.
        //    If there is no matching soda name, return false.









}
int main()
{
        VM vm(5);
        vm.restock("Coke",4);
        vm.restock("Diet Coke",5);
        vm.restock("Sprite",1);
        vm.restock("Pepsi",0);
        vm.restock("Lemonade",1);

        if(vm.buySoda("Coke"))
        cout << "I bought " << vm.getSoda("Coke")->getName() << endl;
        else
        cout << "Coke is sold out!!" << endl;

        if(vm.buySoda("Pepsi"))
        cout << "I bought " << vm.getSoda("Pepsi")->getName() << endl;
        else
        cout << "Pepsi is sold out!!" << endl;

        return 0;
}
```

```
bool VM::buySoda(string name)
{
        // TODO: (15 points)
        // 1. Search through all Soda objects to find matching Soda by name
        //    If there is a matching soda by name, and if the quantity is > 0.
        //    then we decrease quantity for the Soda by 1 and return true
        //    If there is a matching soda by name, but the quantity is <= 0
        //    then we return false.
        //    If there is no matching soda name, return false.

        for(int i=0;i<numSoda;i++)
                if( inventory[i]->getName() == name && quantity[i] > 0)
                {
                        quantity[i]--;
                        return true;
                }

                return false;
}
int main()
{
        VM vm(5);
        vm.restock("Coke",4);
        vm.restock("Diet Coke",5);
        vm.restock("Sprite",1);
        vm.restock("Pepsi",0);
        vm.restock("Lemonade",1);

        if(vm.buySoda("Coke"))
        cout << "I bought " << vm.getSoda("Coke")->getName() << endl;
        else
        cout << "Coke is sold out!!" << endl;

        if(vm.buySoda("Pepsi"))
        cout << "I bought " << vm.getSoda("Pepsi")->getName() << endl;
        else
        cout << "Pepsi is sold out!!" << endl;

        return 0;
}
```

---

Problem #2: Given two strings str1 and str2, str1 is the permutation of str2 if all the characters in str1 appear in str2 but in different order. For example, "12345" is the permutation of "54132". Assume that only '1' – '9' will appear in the string. A student coded the following program to solve this problem, but there are something wrong. Please find all the bugs in this program.

---

```
#include <iostream>
#include <string>
using namespace std;

bool isPermutation(string str1, string str2)
{
        if(str1.size() != str2.size())
                return false;

        int i, j, counts[10];

        for(i=0;i<10;i++)
                counts[i] = 0;

        for(i=0;i<str1.size();i++)
                counts[ str1[i] -   0  ] ++; // (1)

        for(i=0;i<str1.size();i++)
                counts[ str2[i] -   0  ] --;   // (2)

        for(i=0;i<10;i++)                     // (3)
                if(counts[i] != 0)            // (4)
                        return true;          // (5)
        return false;                         // (6)
}
```

---

Problem #2: Given two strings str1 and str2, str1 is the permutation of str2 if all the characters in str1 appear in str2 but in different order. For example, "12345" is the permutation of "54132". Assume that only '1' – '9' will appear in the string. A student coded the following program to solve this problem, but there are something wrong. Please find all the bugs in this program. (C)

---

```
#include <iostream>
#include <string>
using namespace std;

bool isPermutation(string str1, string str2)
{
        if(str1.size() != str2.size())
                return false;

        int i, j, counts[10];

        for(i=0;i<10;i++)
                counts[i] = 0;

        for(i=0;i<str1.size();i++)
                counts[ str1[i] -   0  ] ++; // (1) Should be counts[str1[i]- '0']++;

        for(i=0;i<str1.size();i++)
                counts[ str2[i] -   0  ] --;  // (2) Should be counts[str2[i]- '0']--;

        for(i=0;i<10;i++)                    // (3)
                if(counts[i] != 0)           // (4)
                        return true;         // (5) Should be return false;
        return false;                        // (6) Should be return true;
}
```

```
int main()
{
        cout << isPermutation("12345","54321") << endl;
        cout << isPermutation("12345","98765") << endl;
        return 0;
}
```

The bugs are in?

(A)123456
(B)12356
(C)1256
(D)234
(E)2356
(F)256
(G)356
(H)3
(I) 12
(J) 34

```
int main()
{
        cout << isPermutation("12345","54321") << endl;
        cout << isPermutation("12345","98765") << endl;
        return 0;
}
```

The bugs are in?

(A) 123456
(B) 12356
(C) 1256
(D) 234
(E) 2356
(F) 256
(G) 356
(H) 3
(I) 12
(J) 34

---

Problem #3: A student implemented a function called deleteB(char *msg) to delete all the B in a C-string. Can you please help evaluate his codes below and tell him what is the output of this program?

---

```
#include <iostream>
using namespace std;
void deleteB(char *msg)
{
	char *ptr;
	while(*msg != 0)
	{
		if(*msg =='B' || *msg =='b')
		{
			ptr = msg;

			while (*ptr != 0)
			{
				*ptr=*( ptr + 1 );
				ptr++;
			}
		}
		msg++;
	}
}

int main()
{
	char msg[100] = "BaabbaBaB.";
	deleteB(msg);
	cout << msg;
}
```

(1) aaaa
(2) aabaa
(3) BaaaaB
(4) BabaBaB
(5) aabbaa
(6) aaaaB
(7) Baaaa
(8) aaaBa

---

Problem #3: A student implemented a function called deleteB(char *msg) to delete all the B in a C-string. Can you please help evaluate his codes below and tell him what is the output of this program? (2)

---

```
#include <iostream>
using namespace std;
void deleteB(char *msg)
{
	char *ptr;
	while(*msg != 0)
	{
		if(*msg =='B' || *msg =='b')
		{
			ptr = msg;

			while (*ptr != 0)
			{
				*ptr=*( ptr + 1 );
				ptr++;
			}
		}
		msg++;
	}
}

int main()
{
	char msg[100] = "BaabbaBaB.";
	deleteB(msg);
	cout << msg;
}
```

(1) aaaa.
(2) aabaa.
(3) BaaaaB.
(4) BabaBaB.
(5) aabbaa.
(6) aaaaB.
(7) Baaaa.
(8) aaaBa.

Problem #4: What is the output of the program below?

```
#include <iostream>
#include <string>
using namespace std;
int foo( string s )
{
        if (s.size () < 1)
                return -1;

        int mult = 1;
        int offs = 0;
        switch (s[0])
        {
                case '-':
                        mult = -1;
                        break ;
                case '+':
                        mult = 1;
                        break ;
                default :
                        offs = 1;
                        break ;
        }
        return mult * (s[1 - offs ] - '0');
}

int foo2(string exp)
{
        string exp1="", exp2="", exp3="";
        int i;

        for(i=0;i<=1;i++) exp1 += exp[i];
        for(i=5;i<=7;i++) exp2 += exp[i];
        for(i=8;i<=13;i++) exp3 += exp[i];

        return foo(exp1) + foo(exp2) + foo(exp3);
}
```

```
int main()
{
        cout << foo2("-3+3+555+99999") << endl;
        return 0;
}
```

(A)1
(B)2
(C)3
(D)4
(E)5
(F)6
(G)7
(H)8
(I) 9
(J) 10
(K)11
(L)12

---

Problem #5: Below is a zurt class definition. In the main function, we declare a const Zurt * pointer pointing to a zurt object. At the <BLANK> below, which member functions can we call without causing compilation error? For example, can we use zp->health(); ? There are more than one answers to this question. Please find all the answers.

---

```
#include <iostream>
using namespace std;
class Zurt
{
public:
        Zurt() { m_health = m_row = m_col = 0; }
        Zurt(int health, int r,int c) {
                m_health = health;
                m_row = r;  m_col = c;
        }
        int health() const { return m_health; }
        int row() const { return m_row; }
        int col() const { return m_col; }
        void setHealth(int health) { m_health = health; }
        void setRow(int r) { m_row = r; }
        void setCol(int c) { m_col = c; }
private:
        int m_health, m_row, m_col;
};

int main()
{
        Zurt z(100,1,1);
        const Zurt* zp = &z;

        <BLANK>

        return 0;
}
```

(A) zp->Zurt(); (B) zp->Zurt(100,1,2); (C) int h = zp->health();
(D) int r = zp->row(); (E) int c = zp->col(); (F) zp->setHealth(50);
(G) zp-> setRow(2); (H) zp->setCol(2); (I) zp->m_col = 5;

---

Problem #6: Below is a zurt class definition. In the main function, we declare a Zurt * pointer pointing to a zurt object. At the <BLANK> below, which member functions can we call without causing compilation error? For example, can we use zp->health(); ? There are more than one answers to this question. Please find all the answers.

---

```
#include <iostream>
using namespace std;
class Zurt
{
public:
        Zurt() { m_health = m_row = m_col = 0; }
        Zurt(int health, int r,int c) {
                m_health = health;
                m_row = r;  m_col = c;
        }
        int health() const { return m_health; }
        int row() const { return m_row; }
        int col() const { return m_col; }
        void setHealth(int health) { m_health = health; }
        void setRow(int r) { m_row = r; }
        void setCol(int c) { m_col = c; }
private:
        int m_health, m_row, m_col;
};

int main()
{
        Zurt z(100,1,1);
        Zurt* zp = &z;

        <BLANK>

        return 0;
}
```

(A) zp->Zurt(); (B) zp->Zurt(100,1,2); (C) int h = zp->health();
(D) int r = zp->row(); (E) int c = zp->col(); (F) zp->setHealth(50);
(G) zp-> setRow(2); (H) zp->setCol(2); (I) zp->m_col = 5;

---

Problem #7: What is the output of the program below?

---

```
#include <iostream>
using namespace std;
int main()
{
        int arr[12] = {1,3,5,0,7,2,0,4,4,0,8,8};
        int count = 0;
        for(int i=0;i<11;i++) {
                if(arr[i] = arr[i+1] )
                        count++;
                else
                        count--;
        }
        cout << count << endl;
}
```

---

Problem #8: What is the output of the program below?

---

```
#include <iostream>
using namespace std;
int main()
{
        int arr[100] = {1,1,2,3,5,8};
        int *p = (arr+6);
        for(int i=0;i<2;i++) {
                *p = *(p-1) + *(p-2);
                p++;
        }
        cout << *(p-1) << endl;
}
```

---

Problem #9: What is the output of the program below?

---

```
#include <iostream>
using namespace std;
void swap2(int* a, int *b)
{
   int temp = *a;   *a = *b;   *b = temp;
}
int main()
{
        int arr[7] = { 2,4,6,8,1,3,5 };
        int* first = arr;
        int* last = arr+6;
        int divider = 4;

        while(first < last) {
                while( (first < last) && (*first < divider) )  first++;
                while( (first < last) && (*last  > divider) )  last--;
                swap2(first,last);
        }
        for(first = arr ; first < arr+7; first++)
                cout << *first << " ";
        cout << endl;
}
```

1. Assume the following variable declarations:

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

Which of the following statements will change the value of `foo` to 1?

(a) `ptr++;` (b) `foo++;` (c) `(*foo)++;` (d) `(*ptr)++;`

(e) (a) and (b) only (f) (a) and (d) only (g) (b) and (d) only (e) (c) and (d) only

1. Assume the following variable declarations:

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

Which of the following statements will change the value of `foo` to 1?

(a) `ptr++;` (b) `foo++;` (c) `(*foo)++;` (d) `(*ptr)++;`

(e) (a) and (b) only (f) (a) and (d) only **(g) (b) and (d) only** (e) (c) and (d) only

2. What is the output of the following code segment if the input value is the last digit of your student ID?

```
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;
```

2. What is the output of the following code segment if the input value is the last digit of your student ID?

```
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;
```

| input | output |
|---|---|
| 0 | 6 |
| 1 | 3 |
| 2 | 6 |
| 3 | -9 |
| 4 | 2 |
| 5 | -7 |
| 6 | 4 |
| 7 | 2 |
| 8 | -1 |
| 9 | -7 |

4. Your high school friend who didn't make it to UCLA and settled for a university called U$C sent you the following message through Facebook:

> Hey, I wrote the following function, `countMatches`, which is supposed to compare two C strings and count the number of matching characters. Two characters match if they are the same and if they appear in the same position in each string. For example,
>
> `countMatches("UCLA", "U$C", count);` should set `count` to 1,
> `countMatches("Baseball", "ballpark", count);` should set `count` to 2, etc.
>
> I'm not supposed to create any local variable or square brackets, so I ended up submitting the following code:

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

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

> But it doesn't work! Darn pointers! I asked my friends here at U$C and none of them know how to solve this. Can you tell me what I did wrong, like you always did in high school?

You are too busy preparing for your finals, so you decided to copy and paste his message and just mark the corrections. What is your message going to be? (Make the corrections above.)

4. Your high school friend who didn't make it to UCLA and settled for a university called U$C sent you the following message through Facebook:

> Hey, I wrote the following function, `countMatches`, which is supposed to compare two C-strings and count the number of matching characters. Two characters match if they are the same and if they appear in the same position in each string. For example,
>
> `countMatches("UCLA", "U$C", count);` should set `count` to 1,
> `countMatches("Baseball", "ballpark", count);` should set `count` to 2, etc.
>
> I'm not supposed to create any local variable or square brackets, so I ended up submitting the following code:

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

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

> But it doesn't work! Darn pointers! I asked my friends here at U$C and none of them know how to solve this. Can you tell me what I did wrong, like you always did in high school?

You are too busy preparing for your finals, so you decided to copy and paste his message and just mark the corrections. What is your message going to be? (Make the corrections above).

5. Design the class `Goldfish`, which models a creature that is intelligent enough to remember `capacity` characters at a time.

```
class Goldfish
{
     public:
          Goldfish(int capacity);

          void remember(char c);
          void forget();           // Clears m_memory using dots('.')
          void printMemory() const; // Prints the content of m_memory
     private:
          char m_memory[10];      // memory
          int m_amount;          // # of chars remembered.
          int m_capacity;        // # of chars this fish can remember.
};
```

(a) Define the constructor. Initialize `m_memory` with dot(`'.'`)s. If capacity is not positive, then set it to 3 by default. Remember to set `m_capacity`. If capacity is greater than 10, set it to 10.

```
Goldfish::Goldfish(int capacity)
{
    if (capacity < 1)
        m_capacity = 3;
    else if (capacity > 10)
        m_capacity = 10;
    else
        m_capacity = capacity;
    m_amount = 0;                    // Initialize the amount of memory used.
    forget();                        // Fill in dots.
}
```

To clarify, here is how a `Goldfish` object can be used:

```
int main()
{
      Goldfish nemo(3);
      nemo.remember('a');
      nemo.printMemory();       // prints "a.."
      nemo.remember('b');
      nemo.remember('c');
      nemo.printMemory();       // prints "abc"
      nemo.remember('d');
      nemo.printMemory();       // prints "bcd"
      nemo.forget();
      nemo.printMemory();       // prints "..."
      return 0;
}
```

(b) Implement remember. Store the character `c` into `m_memory`. If you already have `m_capacity` characters memorized, then discard the oldest character in `m_memory` to open up a free slot. (This is an example of LRU (Least Recently Used) replacement.)

```
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++;
}
```

(c) Implement `forget`. Clear memory by filling dot(`'.'`)s into `memory`.

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

6. We'll define Aquarium class, where Goldfish can live.

```
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.
};
```

(a) Define the constructor. Initially, there is no fish in the Aquarium.

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

OR

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

(b) Implement `addFish`, which (dynamically) adds a new `Goldfish` into the `Aquarium`, with the specified memory capacity. If the fish cannot be added because the `Aquarium` already has `MAX_FISH`-many fish residing, return false and don't add any. Otherwise, return true.

(b) Implement `addFish`, which (dynamically) adds a new `Goldfish` into the `Aquarium`, with the specified memory capacity. If the fish cannot be added because the `Aquarium` already has `MAX_FISH`-many fish residing, return false and don't add any. Otherwise, return true.

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

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

(c) Implement `getFish`, which returns the pointer to the `Goldfish` at position n. Return nullptr if there are fewer than `n` fish in the `Aquarium`, or if `n` is an invalid position.

(c) Implement `getFish`, which returns the pointer to the `Goldfish` at position n. Return nullptr if there are fewer than n fish in the `Aquarium`, or if n is an invalid position.

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

    return m_fish[n];
}
```

(d) Implement the destructor. Remove all dynamically allocated objects.

(d) Implement the destructor. Remove all dynamically allocated objects.

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

Make sure you understand why you can't do "**`delete[] m_fish;`**".

(e) Implement `oracle`. It prints the memory of ALL `Goldfish` in the `Aquarium`. You can use the member function `printMemory` of `Goldfish` class for this purpose. Once everything is printed, reset (i.e. clear) all Goldfish's memories using `forget` function.

(e) Implement `oracle`. It prints the memory of ALL `Goldfish` in the `Aquarium`. You can use the member function `printMemory` of `Goldfish` class for this purpose. Once everything is printed, reset (i.e. clear) all Goldfish's memories using `forget` function.

```
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()`**, or **`m_fish[i].forget()`**? That is, why would using dots(**`.`**), instead of arrows (**`->`**), be wrong?

⚠ Will the following code compile? Is there any undefined behavior? If "No" to both questions, what will it print out?

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

struct builder {
    int yearsExperience,
        yearsAtCompany;
    string name;

    // Interview questions were getting stale...
    char favoriteLetter;

    // Get it? Constructor?
    builder () {
        // Must have 2 years experience to apply
        int yearsExperience = 2;
        int yearsAtCompany = 0;
        string name = "Bob";
        favoriteLetter = 'B';
    }
};

int main () {
    builder bob;

    cout << bob.name << endl;
    cout << ((bob.yearsExperience < 2) ? "Veteran" : "Noobuilder") << endl;
}
```

⚠ Will the following code compile? Is there any undefined behavior? If "No" to both questions, what will it print out?

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

struct builder {
    int yearsExperience,
        yearsAtCompany;
    string name;

    // Interview questions were getting stale...
    char favoriteLetter;

    // Get it? Constructor?
    builder () {
        // Must have 2 years experience to apply
        int yearsExperience = 2;
        int yearsAtCompany = 0;
        string name = "Bob";
        favoriteLetter = 'B';
    }
};

int main () {
    builder bob;

    cout << bob.name << endl;
    cout << ((bob.yearsExperience < 2) ? "Veteran" : "Noobuilder") << endl;
}
```

Click for answer.

**Undefined behavior!** bob.yearsExperience is undefined. Look at the constructor: all of these are local variables being declared, and do not refer to the data members! Tricky!

⚠ Will the following code compile? Is there any undefined behavior? If "No" to both questions, what will it print out?

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

struct builder {
    int yearsExperience,
        yearsAtCompany;
    string name;
    char favoriteLetter;

    builder (int experience, string name) {
        // Must have 2 years experience to apply
        int yearsExperience = experience;
        int yearsAtCompany = 0;
        string name = name;
        favoriteLetter = 'B';
    }

    builder (int atCompany, string name) {
        // Must have 2 years experience to apply
        int yearsExperience = 0;
        int yearsAtCompany = atCompany;
        string name = name;
        favoriteLetter = 'B';
    }
};

int main () {
    int experience = 5;
    string name = "Bob";
    builder bob(experience, name);

    cout << bob.name << endl;
    cout << ((bob.yearsExperience < 2) ? "Veteran" : "Noobuilder") << endl;
}
```

⚠ Will the following code compile? Is there any undefined behavior? If "No" to both questions, what will it print out?

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

struct builder {
    int yearsExperience,
        yearsAtCompany;
    string name;
    char favoriteLetter;

    builder (int experience, string name) {
        // Must have 2 years experience to apply
        int yearsExperience = experience;
        int yearsAtCompany = 0;
        string name = name;
        favoriteLetter = 'B';
    }

    builder (int atCompany, string name) {
        // Must have 2 years experience to apply
        int yearsExperience = 0;
        int yearsAtCompany = atCompany;
        string name = name;
        favoriteLetter = 'B';
    }
};

int main () {
    int experience = 5;
    string name = "Bob";
    builder bob(experience, name);

    cout << bob.name << endl;
    cout << ((bob.yearsExperience < 2) ? "Veteran" : "Noobuilder") << endl;
}
```

Click for answer.

**Compile time error!** We can't have two constructors with the same parameter-types, -orders, and -counts. Our poor compilers will get confused :(

⚠ Find all of the syntax errors in the following code:

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

const int NAME_LEN = 100;

class Spy {
    int agentId;
    char name[NAME_LEN];
    bool tellsTruth;
    Spy* target;
    Spy* contact;

    Spy (int id, char codeName[], bool truthiness) {
        agentId = id;
        name = codeName;
        tellsTruth = truthiness;
    }
};

int main () {
    Spy jamesBond(007, "Bond... James Bond", 1);

    cout << jamesBond.name << endl;
    cout << jamesBond.tellsTruth << endl;
}
```

⚠ Find all of the syntax errors in the following code:

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

const int NAME_LEN = 100;

class Spy {
    int agentId;
    char name[NAME_LEN];
    bool tellsTruth;
    Spy* target;
    Spy* contact;

    Spy (int id, char codeName[], bool truthiness) {
        agentId = id;
        name = codeName;
        tellsTruth = truthiness;
    }
};

int main () {
    Spy jamesBond(007, "Bond... James Bond", 1);

    cout << jamesBond.name << endl;
    cout << jamesBond.tellsTruth << endl;
}
```

```
// 1 ---------------------------------
// Constructor is private! We can't
// even make Spy objects
// Fixed:
public:
   Spy (int id, char codeName[], bool truthiness) { ... }

// 2 ---------------------------------
// In constructor: name and codeName
// are cstrings, so we must use strcpy
name = codeName;
// Fixed:
strcpy(name, codeName);

// 3, 4 -----------------------------
// Cannot access private members in the
// cout statements in main!
cout << jamesBond.name << endl;
cout << jamesBond.tellsTruth << endl;
// To fix it, you can either make the
// data members public or make getter
// functions that are public and
// restrict access
```

```cpp
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

const int NAME_LEN = 100;

class Spy {
    int agentId;
    char name[NAME_LEN];
    bool tellsTruth;
    Spy* target;
    Spy* contact;

public:
    Spy (int id, char codeName[], bool truthiness) {
        agentId = id;
        strcpy(name, codeName);
        tellsTruth = truthiness;
        target = nullptr;
        contact = nullptr;
    }

    void setContact (Spy* con) {
        // Betrayal most foul!
        if (contact != nullptr) {
            target = contact;
        }
        contact = con;
    }

    void getTargetFromContact () {
        this->setTarget(contact->target);
    }

    void setTarget (Spy* tar) {
        // Yet more betrayal!
        if (tar == contact) {
            tellsTruth = !tellsTruth;
        }
        target = tar;
    }

    void giveName () {
        cout << ((tellsTruth) ? name : "Timothy Dalton") << endl;
    }

    Spy* getTarget () {
        return target;
    }

    Spy* getContact () {
        return contact;
    }
};
```

⚠ Using the above class definition, find all of the syntax / runtime errors in the f

```cpp
int main () {
    Spy jamesBond(007, "Bond... James Bond", 1);
    Spy blofeld(100, "Blofeld", 0);

    jamesBond.setTarget(blofeld);
    blofeld.setTarget(jamesBond);
}
```

```cpp
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

const int NAME_LEN = 100;

class Spy {
    int agentId;
    char name[NAME_LEN];
    bool tellsTruth;
    Spy* target;
    Spy* contact;

public:
    Spy (int id, char codeName[], bool truthiness) {
        agentId = id;
        strcpy(name, codeName);
        tellsTruth = truthiness;
        target = nullptr;
        contact = nullptr;
    }

    void setContact (Spy* con) {
        // Betrayal most foul!
        if (contact != nullptr) {
            target = contact;
        }
        contact = con;
    }

    void getTargetFromContact () {
        this->setTarget(contact->target);
    }

    void setTarget (Spy* tar) {
        // Yet more betrayal!
        if (tar == contact) {
            tellsTruth = !tellsTruth;
        }
        target = tar;
    }

    void giveName () {
        cout << ((tellsTruth) ? name : "Timothy Dalton") << endl;
    }

    Spy* getTarget () {
        return target;
    }

    Spy* getContact () {
        return contact;
    }
};
```

⚠ Using the above class definition, find all of the syntax / runtime errors

```cpp
int main () {
    Spy jamesBond(007, "Bond... James Bond", 1);
    Spy blofeld(100, "Blofeld", 0);

    jamesBond.setTarget(blofeld);
    blofeld.setTarget(jamesBond);
}
```

```cpp
int main () {
    Spy jamesBond(007, "Bond... James Bond", 1);
    Spy blofeld(100, "Blofeld", 0);

    // [X] setTarget takes *pointers* to Spy objects,
    // not Spy objects themselves
    jamesBond.setTarget(blofeld);
    blofeld.setTarget(jamesBond);
}
```

```cpp
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

const int NAME_LEN = 100;

class Spy {
    int agentId;
    char name[NAME_LEN];
    bool tellsTruth;
    Spy* target;
    Spy* contact;

public:
    Spy (int id, char codeName[], bool truthiness) {
        agentId = id;
        strcpy(name, codeName);
        tellsTruth = truthiness;
        target = nullptr;
        contact = nullptr;
    }

    void setContact (Spy* con) {
        // Betrayal most foul!
        if (contact != nullptr) {
            target = contact;
        }
        contact = con;
    }

    void getTargetFromContact () {
        this->setTarget(contact->target);
    }

    void setTarget (Spy* tar) {
        // Yet more betrayal!
        if (tar == contact) {
            tellsTruth = !tellsTruth;
        }
        target = tar;
    }

    void giveName () {
        cout << ((tellsTruth) ? name : "Timothy Dalton") << endl;
    }

    Spy* getTarget () {
        return target;
    }

    Spy* getContact () {
        return contact;
    }
};
```

```cpp
int main () {
    Spy jamesBond(007, "Bond... James Bond"
    Spy blofeld(100, "Blofeld", 0);

    jamesBond.setTarget(&blofeld);

    // Trust no one but yourself!
    jamesBond.setContact(&jamesBond);
    jamesBond.getTargetFromContact();

    jamesBond.getTarget().giveName();
}
```

```cpp
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

const int NAME_LEN = 100;

class Spy {
    int agentId;
    char name[NAME_LEN];
    bool tellsTruth;
    Spy* target;
    Spy* contact;

public:
    Spy (int id, char codeName[], bool truthiness) {
        agentId = id;
        strcpy(name, codeName);
        tellsTruth = truthiness;
        target = nullptr;
        contact = nullptr;
    }

    void setContact (Spy* con) {
        // Betrayal most foul!
        if (contact != nullptr) {
            target = contact;
        }
        contact = con;
    }

    void getTargetFromContact () {
        this->setTarget(contact->target);
    }

    void setTarget (Spy* tar) {
        // Yet more betrayal!
        if (tar == contact) {
            tellsTruth = !tellsTruth;
        }
        target = tar;
    }

    void giveName () {
        cout << ((tellsTruth) ? name : "Timothy Dalton") << endl;
    }

    Spy* getTarget () {
        return target;
    }

    Spy* getContact () {
        return contact;
    }
};
```

```cpp
int main () {
    Spy jamesBond(007, "Bond... James Bond"
    Spy blofeld(100, "Blofeld", 0);

    jamesBond.setTarget(&blofeld);

    // Trust no one but yourself!
    jamesBond.setContact(&jamesBond);
    jamesBond.getTargetFromContact();

    jamesBond.getTarget().giveName();
}
```

```cpp
int main () {
    Spy jamesBond(007, "Bond... James Bond", 1);
    Spy blofeld(100, "Blofeld", 0);

    jamesBond.setTarget(&blofeld);
    jamesBond.setContact(&jamesBond);
    jamesBond.getTargetFromContact();

    // [X] getTarget() returns a pointer to a Spy,
    // which means if we wanted that Spy to give their
    // name, we must use the -> notation, e.g.:
    // jamesBond.getTarget()->giveName();
    jamesBond.getTarget().giveName();
}
```

```cpp
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

const int NAME_LEN = 100;

class Spy {
    int agentId;
    char name[NAME_LEN];
    bool tellsTruth;
    Spy* target;
    Spy* contact;

public:
    Spy (int id, char codeName[], bool truthiness) {
        agentId = id;
        strcpy(name, codeName);
        tellsTruth = truthiness;
        target = nullptr;
        contact = nullptr;
    }

    void setContact (Spy* con) {
        // Betrayal most foul!
        if (contact != nullptr) {
            target = contact;
        }
        contact = con;
    }

    void getTargetFromContact () {
        this->setTarget(contact->target);
    }

    void setTarget (Spy* tar) {
        // Yet more betrayal!
        if (tar == contact) {
            tellsTruth = !tellsTruth;
        }
        target = tar;
    }

    void giveName () {
        cout << ((tellsTruth) ? name : "Timothy Dalton") << endl;
    }

    Spy* getTarget () {
        return target;
    }

    Spy* getContact () {
        return contact;
    }
};
```

## Example

☑ The following main functions compile and run just fine; what will they print out?

```cpp
int main () {
    Spy jamesBond(007, "Bond... James Bond", 1);
    Spy leChiffre(100, "LeChiffre", 0);
    Spy Q('Q', "Q", 1);

    jamesBond.setContact(&Q);
    Q.setTarget(&leChiffre);
    jamesBond.getTargetFromContact();

    jamesBond.getContact()->giveName();
    jamesBond.getTarget()->giveName();
}
```

```cpp
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

const int NAME_LEN = 100;

class Spy {
    int agentId;
    char name[NAME_LEN];
    bool tellsTruth;
    Spy* target;
    Spy* contact;

public:
    Spy (int id, char codeName[], bool truthiness) {
        agentId = id;
        strcpy(name, codeName);
        tellsTruth = truthiness;
        target = nullptr;
        contact = nullptr;
    }

    void setContact (Spy* con) {
        // Betrayal most foul!
        if (contact != nullptr) {
            target = contact;
        }
        contact = con;
    }

    void getTargetFromContact () {
        this->setTarget(contact->target);
    }

    void setTarget (Spy* tar) {
        // Yet more betrayal!
        if (tar == contact) {
            tellsTruth = !tellsTruth;
        }
        target = tar;
    }

    void giveName () {
        cout << ((tellsTruth) ? name : "Timothy Dalton") << endl;
    }

    Spy* getTarget () {
        return target;
    }

    Spy* getContact () {
        return contact;
    }
};
```

## Example

☑ The following main functions compile and run just fine; what will they print out?

```cpp
int main () {
    Spy jamesBond(007, "Bond... James Bond", 1);
    Spy leChiffre(100, "LeChiffre", 0);
    Spy Q('Q', "Q", 1);

    jamesBond.setContact(&Q);
    Q.setTarget(&leChiffre);
    jamesBond.getTargetFromContact();

    jamesBond.getContact()->giveName();
    jamesBond.getTarget()->giveName();
}
```

```
Q
Timothy Dalton
```

```cpp
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

const int NAME_LEN = 100;

class Spy {
    int agentId;
    char name[NAME_LEN];
    bool tellsTruth;
    Spy* target;
    Spy* contact;

public:
    Spy (int id, char codeName[], bool truthiness) {
        agentId = id;
        strcpy(name, codeName);
        tellsTruth = truthiness;
        target = nullptr;
        contact = nullptr;
    }

    void setContact (Spy* con) {
        // Betrayal most foul!
        if (contact != nullptr) {
            target = contact;
        }
        contact = con;
    }

    void getTargetFromContact () {
        this->setTarget(contact->target);
    }

    void setTarget (Spy* tar) {
        // Yet more betrayal!
        if (tar == contact) {
            tellsTruth = !tellsTruth;
        }
        target = tar;
    }

    void giveName () {
        cout << ((tellsTruth) ? name : "Timothy Dalton") << endl;
    }

    Spy* getTarget () {
        return target;
    }

    Spy* getContact () {
        return contact;
    }
};
```

```cpp
int main () {
    Spy jaws(0, "Jaws", 0);

    // He was never the smartest henchman...
    jaws.setTarget(&jaws);
    jaws.setContact(&jaws);
    jaws.getTargetFromContact();

    jaws.getTarget()->giveName();
}
```

```cpp
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

const int NAME_LEN = 100;

class Spy {
    int agentId;
    char name[NAME_LEN];
    bool tellsTruth;
    Spy* target;
    Spy* contact;

public:
    Spy (int id, char codeName[], bool truthiness) {
        agentId = id;
        strcpy(name, codeName);
        tellsTruth = truthiness;
        target = nullptr;
        contact = nullptr;
    }

    void setContact (Spy* con) {
        // Betrayal most foul!
        if (contact != nullptr) {
            target = contact;
        }
        contact = con;
    }

    void getTargetFromContact () {
        this->setTarget(contact->target);
    }

    void setTarget (Spy* tar) {
        // Yet more betrayal!
        if (tar == contact) {
            tellsTruth = !tellsTruth;
        }
        target = tar;
    }

    void giveName () {
        cout << ((tellsTruth) ? name : "Timothy Dalton") << endl;
    }

    Spy* getTarget () {
        return target;
    }

    Spy* getContact () {
        return contact;
    }
};
```

```cpp
int main () {
    Spy jaws(0, "Jaws", 0);

    // He was never the smartest henchman...
    jaws.setTarget(&jaws);
    jaws.setContact(&jaws);
    jaws.getTargetFromContact();

    jaws.getTarget()->giveName();
}
```

```
Jaws
```

```cpp
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

const int NAME_LEN = 100;

class Spy {
    int agentId;
    char name[NAME_LEN];
    bool tellsTruth;
    Spy* target;
    Spy* contact;

public:
    Spy (int id, char codeName[], bool truthiness) {
        agentId = id;
        strcpy(name, codeName);
        tellsTruth = truthiness;
        target = nullptr;
        contact = nullptr;
    }

    void setContact (Spy* con) {
        // Betrayal most foul!
        if (contact != nullptr) {
            target = contact;
        }
        contact = con;
    }

    void getTargetFromContact () {
        this->setTarget(contact->target);
    }

    void setTarget (Spy* tar) {
        // Yet more betrayal!
        if (tar == contact) {
            tellsTruth = !tellsTruth;
        }
        target = tar;
    }

    void giveName () {
        cout << ((tellsTruth) ? name : "Timothy Dalton") << endl;
    }

    Spy* getTarget () {
        return target;
    }

    Spy* getContact () {
        return contact;
    }
};
```

```cpp
int main () {
    Spy jamesBond(007, "Bond... James Bond", 1);
    Spy leChiffre(100, "LeChiffre", 0);
    Spy vesper(0, "Vesper", 0);

    // NB: I will not answer for any transgressions against
    // actual James Bond plots... this is my practice final dammit!
    jamesBond.setContact(&leChiffre);
    vesper.setContact(&jamesBond);
    vesper.setTarget(&leChiffre);
    leChiffre.setTarget(&jamesBond);
    jamesBond.setContact(&vesper);

    jamesBond.getTarget()->giveName();
    vesper.getTarget()->getTarget()->giveName();
}
```

```cpp
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

const int NAME_LEN = 100;

class Spy {
    int agentId;
    char name[NAME_LEN];
    bool tellsTruth;
    Spy* target;
    Spy* contact;

public:
    Spy (int id, char codeName[], bool truthiness) {
        agentId = id;
        strcpy(name, codeName);
        tellsTruth = truthiness;
        target = nullptr;
        contact = nullptr;
    }

    void setContact (Spy* con) {
        // Betrayal most foul!
        if (contact != nullptr) {
            target = contact;
        }
        contact = con;
    }

    void getTargetFromContact () {
        this->setTarget(contact->target);
    }

    void setTarget (Spy* tar) {
        // Yet more betrayal!
        if (tar == contact) {
            tellsTruth = !tellsTruth;
        }
        target = tar;
    }

    void giveName () {
        cout << ((tellsTruth) ? name : "Timothy Dalton") << endl;
    }

    Spy* getTarget () {
        return target;
    }

    Spy* getContact () {
        return contact;
    }
};
```

```cpp
int main () {
    Spy jamesBond(007, "Bond... James Bond", 1);
    Spy leChiffre(100, "LeChiffre", 0);
    Spy vesper(0, "Vesper", 0);

    // NB: I will not answer for any transgressions against
    // actual James Bond plots... this is my practice final dammit!
    jamesBond.setContact(&leChiffre);
    vesper.setContact(&jamesBond);
    vesper.setTarget(&leChiffre);
    leChiffre.setTarget(&jamesBond);
    jamesBond.setContact(&vesper);

    jamesBond.getTarget()->giveName();
    vesper.getTarget()->getTarget()->giveName();
}
```

```
Timothy Dalton
Bond... James Bond
```

⚠ Will the following code compile? Does it involve any undefined behavior? If "No" to both questions, what will it print out?

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

struct dullExample {
    int i;

    dullExample (int j) {
        i = j;
    }
};

int main () {
    dullExample* arr[5];

    for (int i = 0; i < 5; i++) {
        arr[i] = new dullExample(i);
    }

    // What is ptr pointing to?
    dullExample* ptr = arr[0];

    for (int j = 0; j < 5; j++) {
        cout << ptr->i << endl;
        ptr++;
    }
}
```

⚠ Will the following code compile? Does it involve any undefined behavior? If "No" to both questions, what will it print out?

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

struct dullExample {
    int i;

    dullExample (int j) {
        i = j;
    }
};

int main () {
    dullExample* arr[5];

    for (int i = 0; i < 5; i++) {
        arr[i] = new dullExample(i);
    }

    // What is ptr pointing to?
    dullExample* ptr = arr[0];

    for (int j = 0; j < 5; j++) {
        cout << ptr->i << endl;
        ptr++;
    }
}
```

**Undefined behavior.** The problem is in our loop where we say:

```
for (int j = 0; j < 5; j++) {
    cout << ptr->i << endl;

    // [X] Problem here: incrementing ptr is wrong because
    // each array element is a pointer to an object *in
    // the heap* (dynamic memory), which are NOT guaranteed
    // to be contiguous addresses
    ptr++;
}
```

Additionally, we do not delete the dynamically allocated dullExample objects. See problem below for the fixes.

**Fix the previous problem to print out the member i of each element of arr. Also, resolve any memory leaks.**

## Fix the previous problem to print out the member i of each element of arr. Also, resolve any memory leaks.

We can fix the code by making sure we access the pointer at each index in arr, and then delete it after we're done:

```
for (int j = 0; j < 5; j++) {
    // [!] Now, ptr will point to the correct objects
    ptr = arr[j];
    cout << ptr->i << endl;

    // [!] Clean up our dynamic allocation along the way
    delete ptr;
}
```

**Example**

The following code compiles and runs with no error. For each statement in the main function, determine what is printed out, taking care to determine what type of object is being printed.

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

const int MAX_LENGTH = 50;
char arr[][MAX_LENGTH] = {
    "ARR",
    "PIRATE",
    "RELATED",
    "JOKE",
    "BECAUSE",
    "ARRRRAY"
};

int main () {
    char* ptr1 = arr[0];
    cout << ptr1 << endl;               // #1

    char* ptr2 = ptr1 + 1;
    cout << ptr2 << endl;               // #2
    cout << ptr2[0] << endl;            // #3

    char* ptr3 = *(arr + 2);
    cout << *(ptr3 + 2) << endl;        // #4
    cout << (ptr3 < ptr2) << endl;      // #5
    cout << (ptr3[7] == '\0') << endl;  // #6

    strcpy(ptr1, ptr3 + 2);
    cout << ptr1 << endl;               // #7
    cout << *ptr1 << endl;              // #8
}
```

**Example**

The following code compiles and runs with no error. For each statement in the main function, determine what is printed out, taking care to determine what type of object is being printed.

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;

const int MAX_LENGTH = 50;
char arr[][MAX_LENGTH] = {
    "ARR",
    "PIRATE",
    "RELATED",
    "JOKE",
    "BECAUSE",
    "ARRRRAY"
};

int main () {
    char* ptr1 = arr[0];
    cout << ptr1 << endl;               // #1

    char* ptr2 = ptr1 + 1;
    cout << ptr2 << endl;               // #2
    cout << ptr2[0] << endl;            // #3

    char* ptr3 = *(arr + 2);
    cout << *(ptr3 + 2) << endl;        // #4
    cout << (ptr3 < ptr2) << endl;      // #5
    cout << (ptr3[7] == '\0') << endl;  // #6

    strcpy(ptr1, ptr3 + 2);
    cout << ptr1 << endl;               // #7
    cout << *ptr1 << endl;              // #8
}
```

1. ARR
2. RR
3. R
4. L
5. 0
6. 1
7. LATED
8. L

```
const int MAX_OPS = 100;

struct opString {
    double d;
    char op[MAX_OPS];

    opString (double start, char* ops) {
        d = start;
        strcpy(op, ops);
    }

    double applyOp () {
        int len = strlen(op);
        if (len == 0) {
            return d;
        }

        for (int i = 0; i < len; i++) {
            char currentOp = op[i];
            switch (currentOp) {
                case '+':
                    d += d;
                    break;
                case '-':
                    d -= d;
                    break;
                case '^':
                    // cmath power function that raises the first
                    // argument to the power of the second and returns
                    // that value
                    d = pow(d, d);
                    break;
                case '*':
                    d *= d;
                    break;
                case '/':
                    d /= d;
                    break;
                default:
                    d++;
                    break;
            }
        }
        return d;
    }
};
```

For each of the following main function snippets, determine whether, for any x and the given string of operations, any of the following are tr

- Can the output ever be negative?
- Can the output ever be positive?
- Can the output ever be zero?

```
int main () {
    double x = ...;
    opString op1(x, "+++");
    cout << op1.applyOp() << endl;
}
```

```
const int MAX_OPS = 100;

struct opString {
    double d;
    char op[MAX_OPS];

    opString (double start, char* ops) {
        d = start;
        strcpy(op, ops);
    }

    double applyOp () {
        int len = strlen(op);
        if (len == 0) {
            return d;
        }

        for (int i = 0; i < len; i++) {
            char currentOp = op[i];
            switch (currentOp) {
                case '+':
                    d += d;
                    break;
                case '-':
                    d -= d;
                    break;
                case '^':
                    // cmath power function that raises the first
                    // argument to the power of the second and returns
                    // that value
                    d = pow(d, d);
                    break;
                case '*':
                    d *= d;
                    break;
                case '/':
                    d /= d;
                    break;
                default:
                    d++;
                    break;
            }
        }
        return d;
    }
};
```

For each of the following main function snippets, determine whether, for any x and the given string of operations, any of the following are tr

- Can the output ever be negative?
- Can the output ever be positive?
- Can the output ever be zero?

```
int main () {
    double x = ...;
    opString op1(x, "+++");
    cout << op1.applyOp() << endl;
}
```

**Can be:** negative (x = -1), zero (x = 0), or positive (x = 1)

```
const int MAX_OPS = 100;

struct opString {
    double d;
    char op[MAX_OPS];

    opString (double start, char* ops) {
        d = start;
        strcpy(op, ops);
    }

    double applyOp () {
        int len = strlen(op);
        if (len == 0) {
            return d;
        }

        for (int i = 0; i < len; i++) {
            char currentOp = op[i];
            switch (currentOp) {
                case '+':
                    d += d;
                    break;
                case '-':
                    d -= d;
                    break;
                case '^':
                    // cmath power function that raises the first
                    // argument to the power of the second and returns
                    // that value
                    d = pow(d, d);
                    break;
                case '*':
                    d *= d;
                    break;
                case '/':
                    d /= d;
                    break;
                default:
                    d++;
                    break;
            }
        }
        return d;
      }
 };
```

```
int main () {
    double x = ...;
    opString op2(x, "+-+");
    cout << op2.applyOp() << endl;
}
```

```
const int MAX_OPS = 100;

struct opString {
    double d;
    char op[MAX_OPS];

    opString (double start, char* ops) {
        d = start;
        strcpy(op, ops);
    }

    double applyOp () {
        int len = strlen(op);
        if (len == 0) {
            return d;
        }

        for (int i = 0; i < len; i++) {
            char currentOp = op[i];
            switch (currentOp) {
                case '+':
                    d += d;
                    break;
                case '-':
                    d -= d;
                    break;
                case '^':
                    // cmath power function that raises the first
                    // argument to the power of the second and returns
                    // that value
                    d = pow(d, d);
                    break;
                case '*':
                    d *= d;
                    break;
                case '/':
                    d /= d;
                    break;
                default:
                    d++;
                    break;
            }
        }
        return d;
      }
 };
```

```
int main () {
    double x = ...;
    opString op2(x, "+-+");
    cout << op2.applyOp() << endl;
}
```

- **Can be:** zero (x = -1, x = 0, x = 1)

```
const int MAX_OPS = 100;

struct opString {
    double d;
    char op[MAX_OPS];

    opString (double start, char* ops) {
        d = start;
        strcpy(op, ops);
    }

    double applyOp () {
        int len = strlen(op);
        if (len == 0) {
            return d;
        }

        for (int i = 0; i < len; i++) {
            char currentOp = op[i];
            switch (currentOp) {
                case '+':
                    d += d;
                    break;
                case '-':
                    d -= d;
                    break;
                case '^':
                    // cmath power function that raises the first
                    // argument to the power of the second and returns
                    // that value
                    d = pow(d, d);
                    break;
                case '*':
                    d *= d;
                    break;
                case '/':
                    d /= d;
                    break;
                default:
                    d++;
                    break;
            }
        }
        return d;
    }
};
```

```
int main () {
    double x = ...;
    opString op3(x, "+-^$^");
    cout << op3.applyOp() << endl;
}
```

```
const int MAX_OPS = 100;

struct opString {
    double d;
    char op[MAX_OPS];

    opString (double start, char* ops) {
        d = start;
        strcpy(op, ops);
    }

    double applyOp () {
        int len = strlen(op);
        if (len == 0) {
            return d;
        }

        for (int i = 0; i < len; i++) {
            char currentOp = op[i];
            switch (currentOp) {
                case '+':
                    d += d;
                    break;
                case '-':
                    d -= d;
                    break;
                case '^':
                    // cmath power function that raises the first
                    // argument to the power of the second and returns
                    // that value
                    d = pow(d, d);
                    break;
                case '*':
                    d *= d;
                    break;
                case '/':
                    d /= d;
                    break;
                default:
                    d++;
                    break;
            }
        }
        return d;
      }
 };
```

```
int main () {
    double x = ...;
    opString op3(x, "+-^$^");
    cout << op3.applyOp() << endl;
}
```

> **Always strictly positive** (in fact, is 4 for all input; remember that 0 ^ 0 = 1)

```
const int MAX_OPS = 100;

struct opString {
    double d;
    char op[MAX_OPS];

    opString (double start, char* ops) {
        d = start;
        strcpy(op, ops);
    }

    double applyOp () {
        int len = strlen(op);
        if (len == 0) {
            return d;
        }

        for (int i = 0; i < len; i++) {
            char currentOp = op[i];
            switch (currentOp) {
                case '+':
                    d += d;
                    break;
                case '-':
                    d -= d;
                    break;
                case '^':
                    // cmath power function that raises the first
                    // argument to the power of the second and returns
                    // that value
                    d = pow(d, d);
                    break;
                case '*':
                    d *= d;
                    break;
                case '/':
                    d /= d;
                    break;
                default:
                    d++;
                    break;
            }
        }
        return d;
    }
};
```

```
int main () {
    double x = ...;
    opString op4(x, "***");
    cout << op4.applyOp() << endl;
}
```

```
const int MAX_OPS = 100;

struct opString {
    double d;
    char op[MAX_OPS];

    opString (double start, char* ops) {
        d = start;
        strcpy(op, ops);
    }

    double applyOp () {
        int len = strlen(op);
        if (len == 0) {
            return d;
        }

        for (int i = 0; i < len; i++) {
            char currentOp = op[i];
            switch (currentOp) {
                case '+':
                    d += d;
                    break;
                case '-':
                    d -= d;
                    break;
                case '^':
                    // cmath power function that raises the first
                    // argument to the power of the second and returns
                    // that value
                    d = pow(d, d);
                    break;
                case '*':
                    d *= d;
                    break;
                case '/':
                    d /= d;
                    break;
                default:
                    d++;
                    break;
            }
        }
        return d;
    }
};
```

```
int main () {
    double x = ...;
    opString op4(x, "***");
    cout << op4.applyOp() << endl;
}
```

- **Can be:** zero (x = 0) or positive (x = 1)

# Goodbye!

- You're going to do fine, breathe deep, focus on your problems.
- Try not to look at the giant, ticking, countdown clock unless absolutely necessary.
- Start with a question you are sure about, this will give you confidence in tackling the tough ones.
- Skip some question if you know that they will take up a lot of time.
- *FILL YOUR TA EVALUATIONS ON MYUCLA.*

# Goodbye!

• I hope that I did a decent job, and you all are satisfied the way the discussions went.
• I would appreciate any kind of feedback, good or bad, from you all. This would help me improve myself.
• A special thank you to all of you, it has been a pleasure to teach and get to know you all.

All the best for everything, study hard and enjoy life.

# Thank You!