given the following snippet below, how's the affected variable changed? (‘.’’ indicates nothing changed)

```c
void foo(int& x) {
  x++
}

void bar(int* x) {
  x++
  (*x)++;
}

void baz(int x[]) {
  x++
  (*x)++;
}

int main() {
  int x = 2;
  int* px = &x;
  int ax[5] = {5, -4, 3, -2, 1};
  int *pax = ax;
  foo(x);
  x = 3
  (*px)++;
  x = 4
  foo(ax);
  error: ax is an int[] and foo needs int&
  foo(ax[1]);
  ax = {., -3,.,.,.}
  bar(ax);
  ax = {., -2,.,.,.} // int[] can be treated
  as an int* for fxn args
  baz(pax);
  ax = {., -1,.,.,.} // int* can be treated
  as int[] for fxn args
  foo(*pax);
  ax = {6,.,.,.,.}
  bar(pax + 2);
  ax = {.,.,.,-1,.} // bar(ax + 2) will do
  the same
  baz(pax + *px);
  error: pax + 4 is already at the last
  element of ax, and baz walks off the end
  of it with the x++

  // at the end, ignoring erroneous steps:
  // ax = {6, -1, 3, -1, 1}
```
cstring comparison

Implement a function that computes the “edit distance” of two strings. We define the “edit distance” as the number of characters that differ between two strings, where extra characters from the longer string count for a distance of 1 (ignore spaces).

```
dist("hey", "her");  // 1
dist("hey", "he");  // 1
dist("hello", "goodbye");  // 7
dist("am i sam?", "he is american!");  // 9
```
cstring comparison (answer)

Implement a function that computes the “edit distance” of two strings. We define the “edit distance” as the number of characters that differ between two strings, where extra characters from the longer string count for a distance of 1 (ignore spaces).

```c
int dist(char* str1, char* str2) {
    int d = 0;
    while (*str1 != 0 || *str2 != 0) {
        if (*str1 == ' ') {
            str1++;
        } else if (*str2 == ' ') {
            str2++;
        } else {
            if (*str1 != *str2) {
                d++;
            }
            if (*str1 != 0) {
                str1++;
            }
            if (*str2 != 0) {
                str2++;
            }
        }
    }
    return d;
}
```

dist(“hey”, “her”); // 1
dist(“hey”, “he”); // 1
dist(“hello”, “goodbye”); // 7
dist(“am i sam?”, “he is american!”); // 9
Design a color that is constructed with an RGB color value and has methods to darken (-1 to all channels) or lighten (+1 to all channels) the color. The class can also print its RGB values. Implement those functions shown below. (Also, do we need a destructor?)

```cpp
class Color {
private:
    int RGB[3];

public:
    Color(); // inits random color
    Color(int r, int g, int b);
    void showRGB() const;
    void lighten();
    void darken();
};
```
Design a color that is constructed with an RGB color value and has methods to darken (-1 to all channels) or lighten (+1 to all channels) the color. The class can also print its RGB values. Implement those functions shown below. (Also, do we need a destructor?)

Nope, no dynamically allocated data is allocated and stored in a Color object.

```cpp
class Color {
private:
    int RGB[3];

public:
    Color(); // inits random color
    Color(int r, int g, int b);
    void showRGB() const;
    void lighten();
    void darken();
};
```

```cpp
Color::Color() {
    for (int k = 0; k < 3; k++) {
        RGB[k] = rand() % 256;
    }
}

Color::Color(int r, int g, int b) {
    RGB[0] = r; RGB[1] = g; RGB[2] = b;
}

void Color::showRGB() const {
    cout << "r: " << RGB[0] << ", ";
    cout << "g: " << RGB[1] << ", ";
    cout << "b: " << RGB[2] << endl;
}

void Color::lighten() {
    for (int k = 0; k < 3; k++)
        RGB[k]++;
}

void Color::darken() {
    for (int k = 0; k < 3; k++)
        RGB[k]--;
}
```
object classes: p1.2 - color palette

Now let’s make a whole palette of colors. And we can lighten and darken them all to modify the style of our palette. Implement the skeleton below:

```cpp
const int MAX_COLORS = 100;

class ColorPalette {
    private:
        Color* _palette[MAX_COLORS];
        int _ncolors;

    public:
        ColorPalette();
        ~ColorPalette();

        void addColor(int r, int g, int b);
        void removeColor(int index);

        void lighten();
        void darken();

        void showPalette() const;
};
```
object classes: p1.2 - color palette (answer)

Now let's make a whole palette of colors. And we can lighten and darken them all to modify the style of our palette. Implement the skeleton below:

```cpp
const int MAX_COLORS = 100;

class ColorPalette {
private:
    Color* _palette[MAX_COLORS];
    int _ncolors;

public:
    ColorPalette();
    ~ColorPalette();
    void addColor(int r, int g, int b);
    void removeColor(int index);
    void lighten();
    void darken();
    void showPalette() const;
};

void ColorPalette::ColorPalette() {
    _ncolors = 0;
}

ColorPalette::~ColorPalette() {
    for (int k = 0; k < _ncolors; k++) {
        delete _palette[k];
    }
}

void ColorPalette::addColor(int r, int g, int b) {
    if (_ncolors + 1 < MAX_COLORS) {
        _palette[_ncolors] = new Color(r, g, b);
        _ncolors++;
    }
}

void ColorPalette::removeColor(int index) {
    if ((index > 0) && (index < _ncolors)) {
        delete _palette[k];
        _palette[k] = _palette[_ncolors - 1];
        _ncolors--;
    }
}

void ColorPalette::lighten() {
    for (int k = 0; k < _ncolors; k++) {
        _palette[k]->lighten();
    }
}

void ColorPalette::darken() {
    for (int k = 0; k < _ncolors; k++) {
        _palette[k]->darken();
    }
}

void ColorPalette::showPalette() {
    cout << "---ColorPalette---" << endl;
    for (int k = 0; k < _ncolors; k++) {
        _palette[k]->showRGB();
    }
}
```
design a water bottle that keeps track of how much the owner drinks each day for up to 7 days. The tiny LCD display shows the user how much water is being consumed on average over the past week. An example program of the user and bottle follows. Implement the necessary functions.

```cpp
int main() {
    SmartBottle bottle(16); // create an empty bottle with a maximum capacity (oz)
    bottle.add(2); // adds water (in oz)
    cout << bottle.left() << endl; // returns how many oz the bottle has
    bottle.drink(4); // how much water one drinks (no more than in the bottle)
    bottle.nextday(); // starts a new day of recording
    bottle.add(10);
    bottle.drink(4);
    bottle.nextday();
    bottle.add(12); // :( - spilled 2 oz
    bottle.drink(5);
    bottle.add(3);
    bottle.drink(4);
    bottle.drink(4);
    bottle.drank(0); // tells how much was drank today
    bottle.drank(1); // tells how much was drank: 1=yesterday, 2=day before, ...
    bottle.nextday();
    bottle.eject(); // empties the bottle
    bottle.add(2);
    bottle.drink(1);
    if (bottle.consumed_avg() < 10) { // returns average oz consumed per day
        cout << "Drink more water!" << endl;
    }
}
```
object classes: p2 - smart water bottle (guide)

First, what functions do we need to write? Second, figure out what variables you might need (likely an int for the number of oz in the bottle and an array of ints for how much was drank each day over the past 7 days). Lastly, start writing the class.

```cpp
int main() {
    SmartBottle bottle(16); // create an empty bottle with a maximum capacity (oz)
bottle.add(2); // adds water (in oz)
cout << bottle.left() << endl; // returns how many oz the bottle has
bottle.drink(4); // how much water one drinks (no more than in the bottle)
bottle.nextday(); // starts a new day of recording
bottle.add(10);
bottle.drink(4);
bottle.nextday();
bottle.add(12); // :( - spilled 2 oz
bottle.drink(5);
bottle.add(3);
bottle.drink(4);
bottle.drink(4);
bottle.drank(0); // tells how much was drank today
bottle.drank(1); // tells how much was drank: 1=yesterday, 2=day before, ...
bottle.nextday();
bottle.eject(); // empties the bottle
bottle.add(2);
bottle.drink(1);
if (bottle.consumed_avg() < 10) { // returns average oz consumed per day
    cout << “Drink more water!” << endl;
}
```
```cpp
const int NDAYS = 7;

class SmartBottle {
    private:
        int _amount;
        int _drank[NDAYS];
        int _max_amount;
        int _day;
    
    public:
        SmartBottle(int max);
        void add(int n_oz);
        void drink(int n_oz);
        void nextday();
        void eject();
        int left() const;
        int drank(int day) const;
        double consumed_avg();
    
    SmartBottle::SmartBottle(int max) {
        _max_amount = max;
        _amount = 0;
        _day = 0;
        for (int k = 0; k < NDAYS; k++) {
            _drank[k] = 0;
        }
    }
    
    void SmartBottle::add(int noz) {
        if (noz > 0) {
            _amount += noz;
            if (_amount > _max_amount) {
                _amount = max_amount;
            }
        }
    }
    
    void SmartBottle::drink(int noz) {
        if (noz > 0) {
            if (noz > _amount) {
                noz = _amount;
            }
            _amount -= noz;
            _drank[_day] += noz;
        }
    }
    
    void SmartBottle::nextday() {
        _day++;
        if (_day == NDAYS) {
            _day = 0;
        }
        _drank[_day] = 0;
    }
    
    int SmartBottle::left() const {
        return _amount;
    }
    
    int SmartBottle::drank(int n) {
        if ((n > 0) && (n < NDAYS)) {
            int old_day = _day - n;
            if (old_day < 0) {
                old_day += NDAYS;
            }
            return _drank[old_day];
        }
    }
    
    double SmartBottle::consumed_avg() {
        double avg = 0.0;
        for (int k = 0; k < NDAYS; k++) {
            avg += _drank[k];
        }
        return avg / NDAYS;
    }
    
    void SmartBottle::eject() {
        _amount = 0;
    }
};
```
Your smart water bottle was a huge hit! now it’s time to make a business out of it and reap the rewards of your programming genius. Implement a simple store class to sell your (currently tax-free) product, which keeps track of how much you have in stock, how much you’re making, and can be used as shown below:

```cpp
int main() {
  Store myshop;
  myshop.cost(9.00);

  // january
  myshop.price(13.99);
  myshop.get_shipment(100);
  myshop.sell(90);

  // february
  myshop.get_shipment(120);
  myshop.price(15.99);
  myshop.sell(60);

  // march
  myshop.get_shipment(80);
  myshop.price(7.99);
  myshop.sell(150);

  // how're we doing?
  cout << "Profit: " << myshop.net();
}
```
Your smart water bottle was a huge hit! now it’s time to make a business out of it and reap the rewards of your programming genius. Implement a simple store class to sell your (currently tax-free) product, which keeps track of how much you have in stock, how much you’re making, and can be used as shown below:

```cpp
int main() {
    Store myshop;
    myshop.cost(9.00);

    // january
    myshop.price(13.99);
    myshop.get_shipment(100);
    myshop.sell(90);

    // february
    myshop.get_shipment(120);
    myshop.price(15.99);
    myshop.sell(60);

    // march
    myshop.get_shipment(80);
    myshop.price(7.99);
    myshop.sell(150);

    // how’re we doing?
    cout << “Profit: ”;
    cout << myshop.net();
    cout << endl;
}
```

```cpp
class Store {
    private:
        int _nunits;
        double _price;
        double _cost;
        int _totalcost;
        int _totalgain;

    public:
        Store();
        void price(double x);
        void get_shipment(int n);
        void sell(int n);
        double net();
}
```

```cpp
Store::Store() {
    _nunits = 0;
    _price = -1.0; // anything
    _cost = -1.0; // anything
    _totalcost = 0;
    _totalgain = 0;
}
```

```cpp
void Store::sell(int n) {
    if (n > _nunits) {
        n = _nunits;
    }
    _nunits -= n;
    _totalgain += (n * _price);
}
```

```cpp
void Store::price(double x) {
    _price = x;
}
```

```cpp
void Store::get_shipment(int n) {
    _nunits += n;
    _totalcost += (n * _cost);
}
```

```cpp
double Store::net() {
    return _totalgain - _totalcost;
}
```