Project 4 Test Data

There were 84 test cases. The first 64 were worth 1.11 point each; the next 17 were worth .82 points each; the last three, worth one point each, were the bonuses you earned for certain functions that passed all tests but did not use an additional array. To run the test cases:

1. Remove the main routine from your `array.cpp` file.
2. Append the following text to the end of your `array.cpp` file, and build the resulting program.
3. For any test case you wish to try, run the program, providing as input the test number.

```
#include <iostream>
#include <string>
#include <algorithm>
#include <cstdlib>
#include <cassert>

using namespace std;

string c[6] = {
        "alpha", "beta", "beta", "delta", "gamma", "gamma"
};

bool dividecheck(const string a[], int n, int p, string divider)
{
        for (int k = 0; k < p; k++)
                if (a[k] >= divider)
                        return false;
        for ( ; p < n  &&  a[p] == divider; p++)
                ;
        for (int k = p; k < n; k++)
                if (a[k] <= divider)
                        return false;
        string b[100];
        copy(a, a+n, b);
        sort(b, b+n);
        return equal(b, b+n, c);
}

void testone(int n)
{
        const int N = 6;

      // Act as if  a  were declared:
      //   string a[N] = {
      //       "alpha", "beta", "gamma", "gamma", "beta", "delta"
      //   };
      // This is done in a way that will probably cause a crash if
      // a[-1] or a[N] is accessed:  We place garbage in those positions.
        string aa[1+N+1] = {
                "", "alpha", "beta", "gamma", "gamma", "beta", "delta", ""
        };
        string* a = &aa[1];
        string* z = aa;
        a[-1].string::~string();
        a[N].string::~string();
        fill_n(reinterpret_cast<char*>(&a[-1]), sizeof(a[-1]), 0xEF);
        fill_n(reinterpret_cast<char*>(&a[N]), sizeof(a[N]), 0xEF);

        string b[N] = {
                "alpha", "beta", "gamma", "delta", "beta", "delta"
        };

        string d[9] = {
                "alpha", "beta",  "beta", "beta", "alpha",
                "alpha", "delta", "beta", "beta"
        };

        switch (n)
        {
                                 case  1: {
                assert(appendToAll(z, -1, "rho") == -1 && a[0] == "alpha");
                        } break; case  2: {
                assert(appendToAll(z, 0, "rho") == 0 && a[0] == "alpha");
                        } break; case  3: {
                assert(appendToAll(a, 1, "rho") == 1 && a[0] == "alpharho" &&
                                                    a[1] == "beta");
                        } break; case  4: {
                assert(appendToAll(a, 6, "rho") == 6 && a[0] == "alpharho" &&
                        a[1] == "betarho" && a[2] == "gammarho" &&
                        a[3] == "gammarho" && a[4] == "betarho" &&
                        a[5] == "deltarho");
                        } break; case  5: {
                assert(lookup(z, -1, "alpha") == -1);
                        } break; case  6: {
                assert(lookup(z, 0, "alpha") == -1);
                        } break; case  7: {
                assert(lookup(a, 1, "alpha") == 0);
                        } break; case  8: {
                assert(lookup(a, 6, "delta") == 5);
                        } break; case  9: {
                assert(lookup(a, 6, "beta") == 1);
                        } break; case 10: {
                assert(lookup(a, 6, "zeta") == -1);
                        } break; case 11: {
                assert(positionOfMax(z, -1) == -1);
                        } break; case 12: {
                assert(positionOfMax(z, 0) == -1);
                        } break; case 13: {
```

```
        assert(positionOfMax(a, 1) == 0);
                } break; case 14: {
        assert(positionOfMax(a, 3) == 2);
                } break; case 15: {
        assert(positionOfMax(a, 6) == 2);
                } break; case 16: {
        assert(positionOfMax(a+3, 3) == 0);
                } break; case 17: {
        a[0] = "";
        a[1] = " ";
        a[2] = "";
        assert(positionOfMax(a, 3) == 1);
                } break; case 18: {
        assert(rotateLeft(z, -1, 0) == -1 &&
                        a[0] == "alpha" && a[1] == "beta");
                } break; case 19: {
        assert(rotateLeft(a, 6, -1) == -1 &&
                a[0] == "alpha" && a[1] == "beta" && a[2] == "gamma" &&
                a[3] == "gamma" && a[4] == "beta" && a[5] == "delta");
                } break; case 20: {
        assert(rotateLeft(a, 6, 6) == -1 &&
                a[0] == "alpha" && a[1] == "beta" && a[2] == "gamma" &&
                a[3] == "gamma" && a[4] == "beta" && a[5] == "delta");
                } break; case 21: {
        assert(rotateLeft(z, 0, 0) == -1 &&
                        a[0] == "alpha" && a[1] == "beta");
                } break; case 22: {
        assert(rotateLeft(a, 1, 0) == 0 &&
                        a[0] == "alpha" && a[1] == "beta");
                } break; case 23: {
        assert(rotateLeft(a, 6, 0) == 0 &&
                a[0] == "beta" && a[1] == "gamma" && a[2] == "gamma" &&
                a[3] == "beta" && a[4] == "delta" && a[5] == "alpha");
                } break; case 24: {
        assert(rotateLeft(a, 6, 5) == 5 &&
                a[0] == "alpha" && a[1] == "beta" && a[2] == "gamma" &&
                a[3] == "gamma" && a[4] == "beta" && a[5] == "delta");
                } break; case 25: {
        assert(rotateLeft(a, 6, 3) == 3 &&
                a[0] == "alpha" && a[1] == "beta" && a[2] == "gamma" &&
                a[3] == "beta" && a[4] == "delta" && a[5] == "gamma");
                } break; case 26: {
        assert(rotateLeft(a, 5, 3) == 3 &&
                a[0] == "alpha" && a[1] == "beta" && a[2] == "gamma" &&
                a[3] == "beta" && a[4] == "gamma" && a[5] == "delta");
                } break; case 27: {
        assert(countRuns(z, -1) == -1);
                } break; case 28: {
        assert(countRuns(z, 0) == 0);
                } break; case 29: {
        assert(countRuns(a, 1) == 1);
                } break; case 30: {
        assert(countRuns(a, 3) == 3);
                } break; case 31: {
        assert(countRuns(a, 4) == 3);
                } break; case 32: {
        assert(countRuns(a+2, 4) == 3);
                } break; case 33: {
        assert(countRuns(d, 9) == 5);
                } break; case 34: {
        assert(flip(z, -1) == -1 && a[0] == "alpha");
                } break; case 35: {
        assert(flip(z, 0) == 0 && a[0] == "alpha");
                } break; case 36: {
        assert(flip(a, 1) == 1 && a[0] == "alpha" &&
                                a[1] == "beta");
                } break; case 37: {
        assert(flip(a, 2) == 2 && a[0] == "beta" &&
                                a[1] == "alpha" && a[2] == "gamma");
                } break; case 38: {
        assert(flip(a, 5) == 5 && a[0] == "beta" &&
                a[1] == "gamma" && a[2] == "gamma" && a[3] == "beta" &&
                a[4] == "alpha" && a[5] == "delta");
                } break; case 39: {
        a[2] = "zeta";
        assert(flip(a,6) == 6 && a[0] == "delta" && a[1] == "beta" &&
                a[2] == "gamma" && a[3] == "zeta" && a[4] == "beta" &&
                a[5] == "alpha");
                } break; case 40: {
        assert(differ(z, -1, b, 6) == -1);
                } break; case 41: {
        assert(differ(a, 6, z, -1) == -1);
                } break; case 42: {
        assert(differ(z, 0, b, 0) == 0);
                } break; case 43: {
        assert(differ(a, 3, b, 3) == 3);
                } break; case 44: {
        assert(differ(a, 3, b, 2) == 2);
                } break; case 45: {
        assert(differ(a, 2, b, 3) == 2);
                } break; case 46: {
        assert(differ(a, 6, b, 6) == 3);
                } break; case 47: {
        assert(subsequence(z, -1, b, 6) == -1);
                } break; case 48: {
        assert(subsequence(a, 6, z, -1) == -1);
                } break; case 49: {
        assert(subsequence(z, 0, b, 6) == -1);
                } break; case 50: {
        assert(subsequence(a, 6, z, 0) == 0);
                } break; case 51: {
```

```
assert(subsequence(a, 6, b, 1) == 0);
        } break; case 52: {
assert(subsequence(a, 6, b+4, 2) == 4);
        } break; case 53: {
assert(subsequence(a, 6, b+3, 1) == 5);
        } break; case 54: {
assert(subsequence(a, 6, b+3, 2) == -1);
        } break; case 55: {
assert(subsequence(a, 6, b+2, 2) == -1);
        } break; case 56: {
assert(subsequence(a, 6, a, 6) == 0);
        } break; case 57: {
assert(lookupAny(a, 6, z, -1) == -1);
        } break; case 58: {
assert(lookupAny(z, -1, b, 6) == -1);
        } break; case 59: {
assert(lookupAny(z, 0, b, 1) == -1);
        } break; case 60: {
assert(lookupAny(a, 6, z, 0) == -1);
        } break; case 61: {
assert(lookupAny(a, 1, b, 1) == 0);
        } break; case 62: {
assert(lookupAny(a, 6, b+3, 3) == 1);
        } break; case 63: {
assert(lookupAny(a, 4, b+3, 3) == 1);
        } break; case 64: {
assert(lookupAny(a, 2, b+2, 2) == -1);
        } break; case 65: {
assert(divide(z, -1, "beta") == -1 &&
        a[0] == "alpha" && a[1] == "beta" && a[2] == "gamma" &&
        a[3] == "gamma" && a[4] == "beta" && a[5] == "delta");
        } break; case 66: {
assert(divide(z, 0, "beta") == 0 &&
        a[0] == "alpha" && a[1] == "beta" && a[2] == "gamma" &&
        a[3] == "gamma" && a[4] == "beta" && a[5] == "delta");
        } break; case 67: {
assert(divide(a, 1, "aaa") == 0 &&
        a[0] == "alpha" && a[1] == "beta" && a[2] == "gamma" &&
        a[3] == "gamma" && a[4] == "beta" && a[5] == "delta");
        } break; case 68: {
assert(divide(a, 1, "alpha") == 0 &&
        a[0] == "alpha" && a[1] == "beta" && a[2] == "gamma" &&
        a[3] == "gamma" && a[4] == "beta" && a[5] == "delta");
        } break; case 69: {
assert(divide(a, 1, "zeta") == 1 &&
        a[0] == "alpha" && a[1] == "beta" && a[2] == "gamma" &&
        a[3] == "gamma" && a[4] == "beta" && a[5] == "delta");
        } break; case 70: {
assert(divide(a, 2, "aaa") == 0 &&
        dividecheck(a, 2, 0, "aaa") && a[2] == "gamma" &&
        a[3] == "gamma" && a[4] == "beta" && a[5] == "delta");
        } break; case 71: {
assert(divide(a, 2, "alpha") == 0 &&
        dividecheck(a, 2, 0, "alpha") && a[2] == "gamma" &&
        a[3] == "gamma" && a[4] == "beta" && a[5] == "delta");
        } break; case 72: {
assert(divide(a, 2, "beta") == 1 &&
        dividecheck(a, 2, 1, "beta") && a[2] == "gamma" &&
        a[3] == "gamma" && a[4] == "beta" && a[5] == "delta");
        } break; case 73: {
assert(divide(a, 2, "zeta") == 2 &&
        dividecheck(a, 2, 2, "zeta") && a[2] == "gamma" &&
        a[3] == "gamma" && a[4] == "beta" && a[5] == "delta");
        } break; case 74: {
assert(divide(a, 6, "aaa") == 0 && dividecheck(a, 6, 0, "aaa"));
        } break; case 75: {
assert(divide(a, 6, "alpha") == 0 &&
        dividecheck(a, 6, 0, "alpha"));
        } break; case 76: {
assert(divide(a, 6, "beta") == 1 &&
        dividecheck(a, 6, 1, "beta"));
        } break; case 77: {
assert(divide(a, 6, "delta") == 3 &&
        dividecheck(a, 6, 3, "delta"));
        } break; case 78: {
assert(divide(a, 6, "gamma") == 4 &&
        dividecheck(a, 6, 4, "gamma"));
        } break; case 79: {
assert(divide(a, 6, "zeta") == 6 &&
        dividecheck(a, 6, 6, "zeta"));
        } break; case 80: {
a[2] = "mu";
c[5] = "mu";
assert(divide(a, 6, "mu") == 5 && dividecheck(a, 6, 5, "mu"));
        } break; case 81: {
assert(divide(a, 6, "chi") == 3 && dividecheck(a, 6, 3, "chi"));
        } break; case 82: {
  // To earn the bonus point for rotateLeft, this and all other
  // rotateLeft tests must pass, and that function must not
  // use any additional arrays.
const int BIG = 500;
string h[BIG];
for (int k = 0; k < BIG; k++)
        h[k] = (k % 2 == 0 ? "alpha" : "beta");
h[BIG-2] = "gamma";
h[BIG-1] = "delta";
rotateLeft(h, BIG, 0);
assert(h[BIG-3] == "gamma"  &&  h[BIG-2] == "delta");
        } break; case 83: {
  // To earn the bonus point for flip, this and all other
```

```
                  // flip tests must pass, and that function must not
                  // use any additional arrays.
                const int BIG = 500;
                string h[BIG];
                for (int k = 0; k < BIG; k++)
                        h[k] = (k % 2 == 0 ? "alpha" : "beta");
                h[0] = "gamma";
                h[BIG-1] = "delta";
                flip(h, BIG);
                assert(h[0] == "delta"  &&  h[BIG-1] == "gamma");
                        } break; case 84: {
                  // To earn the bonus point for divide, this and all other
                  // divide tests must pass, and that function must not
                  // use any additional arrays.
                const int BIG = 500;
                string h[BIG];
                string i[3] = { "alpha", "beta", "gamma" };
                for (int k = 0; k < BIG; k++)
                        h[k] = i[k % 3];
                divide(h, BIG, "beta");
                int m = 0;
                for (m = 0; m < (BIG+2)/3; m++)
                        assert(h[m] == "alpha");
                for ( ; m < (BIG+1)/3; m++)
                        assert(h[m] == "beta");
                for ( ; m < BIG/3; m++)
                        assert(h[m] == "gamma");
                        } break;
        }

        new (&a[-1]) string;
        new (&a[N]) string;
}

int main()
{
    cout << "Enter a test number (1 to 84): ";
    int n;
    cin >> n;
    if (n < 1  ||  n > 84)
    {
        cout << "Bad test number" << endl;
        return 1;
    }
    testone(n);
    cout << "Passed test " << n << endl;
}
```