**Midterm**

1. In the sport of diving, seven judges award a score between 0 and 10, where each score may be a floating-point value. The highest and lowest scores are thrown out and the remaining scores are added together. The sum is then multiplied by the degree of difficulty for that dive. The degree of difficulty ranges from 1.2 to 3.8 points. The total is then multiplied by 0.6 to determine the diver’s score. Write a computer program that inputs a degree of difficulty and seven judges’ scores, and outputs the overall score for that dive. The program should ensure that all inputs are within the allowable data ranges.  
     
   #include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

const int sizeA = 7;

const float D = 0.6;

float getMin(float[]);

float getMax(float[]);

float getSum(float[]);

void input(float[]);

int main()

{

float scores[sizeA];

float min, max, sum;

float dScore, difficulty;

input(scores);

dScore = getSum(scores);

dScore -= getMin(scores);

dScore -= getMax(scores);

do{

cout << "Enter degree of difficulty ranges from 1.2 to 3.8 points ==> ";

cin >> difficulty;

} while (difficulty<1.2 || difficulty>3.8);

dScore \*= difficulty;

dScore \*= D;

cout << "Diver's Score: " << dScore << endl;

return 0;

}

void input(float a[]) {

srand(time(0));

for (int i = 0; i < sizeA; i++){

a[i] = rand() % 11;

cout << "Judge " << (i+1) << ") Enter a score==> " << a[i] << endl;

}

}

float getMin(float a[]) {

float min = a[0];

for (int i = 1; i < sizeA; i++)

if (a[i] < min)

min = a[i];

return min;

}

float getMax(float a[]) {

float max = a[0];

for (int i = 1; i < sizeA; i++)

if (a[i] > max)

max = a[i];

return max;

}

float getSum(float a[]) {

float sum = 0;

for (int i = 0; i < sizeA; i++)

sum += a[i];

return sum;

}

1. In an ancient land, the beautiful princess Eve had many suitors. She decided on the following procedure to determine which suitor she would marry. First, all of the suitors would be lined up one after the other and assigned numbers. The first suitor would be number 1, the second number 2, and so on up to the last suitor, number *n*. Starting at the first suitor she would then count three suitors down the line (because of the three letters in her name) and the third suitor would be eliminated from winning her hand and removed from the line. Eve would then continue, counting three more suitors, and eliminating every third suitor. When she reached the end of the line she would continue counting from the beginning.

For example, if there were 6 suitors then the elimination process would proceed as follows:

123456 initial list of suitors, start counting from 1

12456 suitor 3 eliminated, continue counting from 4

1245 suitor 6 eliminated, continue counting from 1

125 suitor 4 eliminated, continue counting from 5

15 suitor 2 eliminated, continue counting from 5

1 suitor 5 eliminated, 1 is the lucky winner

Write a program that uses a vector to determine which position you should stand in to marry the princess if there are *n* suitors. You will find the following method from the Vector class useful.

#include <iostream>

#include <vector>

using namespace std;

int main() {

vector<int> v;

int counter;

for (int i = 0; i < 6; i++)

v.push\_back(i + 1);

counter = 0;

while(v.size()!=1)

{

for (int i = 0; i < v.size(); i++)

cout << v[i];

cout << endl;

for (int i = 1; i < 3; i++)

{

if (counter > v.size()-1)

counter = 0;

counter++;

}

v.erase(v.begin() + counter);

}

cout << v[0] << endl << endl;

cout << "The lucky winner is " << v[0] << endl;

return 0;

}

1. This Programming Project requires you to first complete Programming Project 7 from Chapter 5, which is an implementation of a Pizza class. Add an Order class that contains a private vector of type Pizza. This class represents a customer’s entire order, where the order may consist of multiple pizzas. Include appropriate functions so that a user of the Order class can add pizzas to the order (type is deep dish, hand tossed, or pan; size is small, medium, or large; number of pepperoni or cheese toppings). Data members: type and size. Also write a function that outputs everything in the order along with the total price. Write a suitable test program that adds multiple pizzas to an order(s).
2. The bar code on an envelope used by the US Postal Service represents a five (or more) digit zip code using a format called POSTNET (this format is being deprecated in favor of a new system, OneCode, in 2009). The bar code consists of long and short bars as shown below:



For this program we will represent the bar code as a string of digits. The digit 1 represents a long bar and the digit 0 represents a short bar. Therefore, the bar code above would be represented in our program as:

1 10100 10100 01010 11000 01001 1

The first and last digits of the bar code are always 1. Removing these leave 25 digits. If these 25 digits are split into groups of five digits each then we have:

10100 10100 01010 11000 01001

Next, consider each group of five digits. There will always be exactly two 1’s in each group of digits. Each digit stands for a number. From left to right the digits encode the values 7, 4, 2, 1, and 0. Multiply the corresponding value with the digit and compute the sum to get the final encoded digit for the zip code. The table below shows the encoding for 10100.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Bar Code Digits* | *1* | *0* | *1* | *0* | *0* |
| *Value* | *7* | *4* | *2* | *1* | *0* |
| *Product of Digit \* Value* | *7* | *0* | *2* | *0* | *0* |

Zip Code Digit = 7 + 0 + 2 + 0 + 0 = **9**

Repeat this for each group of five digits and concatenate to get the complete zip code. There is one special value. If the sum of a group of five digits is 11, then this represents the digit 0 (this is necessary because with two digits per group it is not possible to represent zero). The zip code for the sample bar code decodes to 99504. While the POSTNET scheme may seem unnecessarily complex, its design allows machines to detect if errors have been made in scanning the zip code.

Write a zip code class that encodes and decodes five digit bar codes used by the US Postal Service on envelopes. The class should have two constructors. The first constructor should input the zip code as an integer and the second constructor should input the zip code as a bar code string consisting of 0’s and 1’s as described above. Although you have two ways to input the zip code, internally the class should only store the zip code using one format (you may choose to store it as a bar code string or as a zip code number.) The class should also have at least two public member functions, one to return the zip code as an integer, and the other to return the zip code in bar code format as a string. All helper functions should be declared private. Embed your class definition in a suitable test program.

class ZipCode

{

public:

ZipCode(int zip);

// Constructs a ZipCode from an integer zip code value

ZipCode(string code);

// Constructs a ZipCode from a string of 1's and 0's

string get\_bar\_code();

// Returns the zip code in bar form

int get\_zip\_code();

// Returns the zip code in numeric form

private:

int zip;

int parse\_bar\_code(string code);

// Returns an integer, parsed from the given bar code.

// If the code is not valid, this function will print

// an error message and then exit.

};

#include <iostream>

#include <string>

using namespace std;

class ZipCode

{

public:

ZipCode(int zip);

// Constructs a ZipCode from an integer zip code value

ZipCode(string code);

// Constructs a ZipCode from a string of 1's and 0's

string get\_bar\_code();

// Returns the zip code in bar form

int get\_zip\_code();

// Returns the zip code in numeric form

private:

int zip;

int parse\_bar\_code(string code);

// Returns an integer, parsed from the given bar code.

// If the code is not valid, this function will print

// an error message and then exit.

};

// Constructs a ZipCode from an integer zip code value

ZipCode::ZipCode(int z) : zip(z)

{

}

// Constructs a ZipCode from a string of 1's and 0's

ZipCode::ZipCode(string code)

{

zip = parse\_bar\_code(code);

}

// Returns an integer, parsed from the given bar code.

// If the code is not valid, this function will print

// an error message and then exit.

int ZipCode::parse\_bar\_code(string code)

{

int value = 0;

//-2 because the first and last 1 of the barcode

//the total number in between should be a multiple of 5

//if not multiple of 5 then the length of the barcode is incorrect

if (((code.length() - 2) % 5) != 0)

{

cout << "ERROR: '" << code << "' has invalid length " << endl

<< " (Length must be 5\*n+2, where n is the number of"

<< endl

<< " digits in the zip code)" << endl;

return -1;

}

//if the first and last character is not a 1

//then error

else if ((code[0] != '1') || (code[code.length() - 1] != '1'))

{

cout << "ERROR: '" << code << "' must begin and end with '1'" << endl;

return -1;

}

//the else means that the length is valid

//the first and last characters are 1

else

{

int digits = (code.length() - 2) / 5;

// Each position 0 - 5 in each 1/0 digit sequence has

// a unique value. Putting these in an array lets us

// map a digit position to a value by doing an array

// lookup, rather than an if-else-if statement.

int values[] = { 7, 4, 2, 1, 0 };

//loops through each group of 5 characters

for (int d = 0; d < digits; d++)

{

int digit = 0;

for (int i = 0; i < 5; i++)

{

char ch = code[d \* 5 + i + 1];

if (ch == '1')

{

digit += values[i];

}

else if (ch != '0')

{

cout << "ERROR: '" << code

<< "' must contain only '1' and '0'" << endl;

// exit(1);

return -1;

}

}

if ((digit < 1) || (digit == 10) || (digit > 11))

{

cout << "ERROR: '" << code << "' has invalid sequence '"

<< code.substr(d \* 5 + 1, 5) << "'" << endl;

// exit(1);

return -1;

}

if (digit == 11)

{

digit = 0;

}

value = (value \* 10) + digit;

}

}

return value;

}

// Returns the zip code in numeric form

int ZipCode::get\_zip\_code()

{

return zip;

}

// Returns the zip code in bar form

string ZipCode::get\_bar\_code()

{

string code;

int z, digit, numDigits, targetDigits;

// Each digit 0-9 has a unique 5-digit bar code sequence.

// If we put these in an array, then we can simply look

// them up via an array index.

string digits[] =

{ "11000", "00011", "00101", "00110", "01001",

"01010", "01100", "10001", "10010", "10100" };

z = zip;

numDigits = 0;

// Work backwards through the zip code integer

while (z > 0)

{

// Pull off the smallest digit

digit = (z % 10);

// Then throw that digit away

z = z / 10;

// Prepend the digit to the code

code = digits[digit] + code;

numDigits++;

}

// Make sure we always have at least 5 or 9 digits (zip or zip+4)

if (numDigits > 5)

{

targetDigits = 9;

}

else

{

targetDigits = 5;

}

while (numDigits < targetDigits)

{

code = digits[0] + code;

numDigits++;

}

// Put a '1' on each end

return "1" + code + "1";

}

// MAIN FUNCTION

int main()

{

ZipCode zip(99504),

zip2("100101010011100001100110001"),

zip3(12345),

zip4(67890);

int BAD\_ZIP\_COUNT = 5;

string bad\_zips[][2] = {

{ "10010101001110000110011001", "bad length" },

{ "000101010011100001100110001", "bad start/end character" },

{ "100101010011100001100110000", "bad start/end character" },

{ "100101010011100001100110021", "bad digit" },

{ "100101010011100001100111001", "bad sequence" }

};

cout << zip.get\_zip\_code() << "'s bar code is '"

<< zip.get\_bar\_code() << "'" << endl;

cout << zip2.get\_zip\_code() << "'s bar code is '"

<< zip2.get\_bar\_code() << "'" << endl;

cout << zip3.get\_zip\_code() << "'s bar code is '"

<< zip3.get\_bar\_code() << "'" << endl;

cout << zip4.get\_zip\_code() << "'s bar code is '"

<< zip4.get\_bar\_code() << "'" << endl;

cout << endl;

// Test a range of values by first constructing a zip code with

// an integer, then retrieving the bar code and using that to

// construct another ZipCode.

int zip\_int = 0;

for (int i = 0; i < 25; i++)

{

// Make an aribrary 5-digit zip code integer, and use it

// to construct a ZipCode

int five\_digit\_zip = (zip\_int \* zip\_int) % 100000;

ZipCode z1(five\_digit\_zip);

// Construct a second ZipCode from the first's bar code

string z1\_code = z1.get\_bar\_code();

ZipCode z2(z1\_code);

cout.width(3);

cout << (i + 1) << ": ";

cout.width(5);

cout << z2.get\_zip\_code() << " has code '"

<< z1\_code << "'";

if ((z1\_code == z2.get\_bar\_code()) &&

(z1.get\_zip\_code() == z2.get\_zip\_code()) &&

(z2.get\_zip\_code() == five\_digit\_zip))

{

cout << " [OK]" << endl;

}

else

{

cout << " [ERR]" << endl;

}

// Increment the test value arbitrarily

zip\_int += (233 + zip\_int % 7);

}

cout << endl;

// Test some error conditions. This test assumes that

// ZipCode will simply set its value to a flag that indicates

// an error, and will not exit the program.

for (int i = 0; i < BAD\_ZIP\_COUNT; i++)

{

cout << "Testing: " << bad\_zips[i][1] << endl;

ZipCode test(bad\_zips[i][0]);

cout << endl;

}

cout << "Enter a character to quit." << endl;

char c;

cin >> c;

return 0;

}