### SWE20004 Technical Software Development Semester 2 2019

# Assignment 1 report

## Name: Jake Scott Student ID: 102581840

## Lab class: Wednesday / 12:30 PM / BA601

**Due Date: Friday 6th September 2019 at 11:59 pm**

**Date Submitted: Friday 6th September 2019 at 05:00 pm**

## Assignment Title: 1

### Program description

This program is used to encode and decode integers, with different rulesets based on how many digits there are in the integer. If a valid integer has been inputted by the user, then the program decides what rule set to apply to the input. Once the integer has been encoded then the program decodes the encoded number to ensure that the number is encoded correctly. Once both encoding and decoding is complete the data is then outputted to the screen.

### Inputs and Outputs

The inputs and outputs for this program are described in Table 1.

**Table 1. Data dictionary:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Data to be stored** | **Sample data** | **Type of data** | **C++ type** | **Input method** | **In / Out** | **Variable name** |
| Input | 2345 | int | Unsigned int | cin | Input | input |
| Loop condition | True | Bool | Bool | declared | n/a | I |
| Input\_convert | 2345 | String | String | input | n/a | Input\_convert |
| Digits | 4 | Int | Unsigned int | Input\_conversion | n/a | digits |
| inputLength | 4 | Int | Int | input | n/a | inputLength |
| First\_digit | 2 | Int | Int | input | n/a | First\_digit |
| Middle\_digit | 3 | Int | Int | input | n/a | Middle\_digit |
| Last\_digit | 4 | Int | Int | Input | n/a | Last\_digit |
| Swapped\_number | 432 | Int | Int | First + second + last digit | n/a | Swapped\_number |
| Array[i] | 2 | Double | Double | input | n/a | Array[i] |
| Product | 2345 | Int | Int | Array[i] | Output | Product |
| secondEncryption | Yes | String | String | cin | Input | secondEncryption |
| Encrypted | Int | Int | Int | input | Cout | Encrypted |
| decodeNumber | Int | Int | Int | encrypted | cout | decodenumber |

### Algorithm

Program steps:

1. Prompt user for input
2. Convert input number to string and get length of number
3. Based on how many digits the number has either accept or reject users input
4. Display “Number contains (x) digits ”
5. Send input variable to the modify function
6. Modify function turns input into string and gets length of number
7. Based on the length of the input determines rule set
8. If number has eight digits, ask user if they want to add second level of encoding
9. Based on rule set user input in encoded and stored in product variable
10. The encoded integer is sent to the decode function
11. Decode function turns encoded data into string then finds length
12. Based on length of encoded integer determines decoding rule set
13. If second level of encoding was applied, second level of decoding begins
14. Decoded data is returned to main function
15. Main function outputs both encoded and decoded data to screen

Encoding and Decoding Logic:

The encoding function represented by f(x) is used to encoding the inputted digits. Below included is the set of inputs that the function accepts.

So given the encoding function f(x) we can see that the inverse of this function is f^-1(x), Therefore we can divide the encrypted output by 3 to find the original input. Below is the set of encrypted numbers from 1 to 10 (left to right).

The pattern here is that they are all multiples of three however only the last number to the multiple is kept so another way of representing these values are:

When the numbers are represented like this the pattern becomes obvious and a simple division of three to each encoded number will inherently decode the data.

After each of the encoded numbers are divided by three, we can see that the inverse function has returned the original input values.

Flow Chart:

A close up of text on a white background

Description automatically generated

:

### Source code:

### #include <iostream>

### #include <cmath>

### #include <sstream>

### #include <string>

### using namespace std;

### string secondEncryption;

### // Function that validates the users input

### int numberInput() {

### unsigned int input;

### string input\_string;

### bool i = true;

### // A while loop that only exits if a valid number is entered.

### while (i) {

### cout << "Enter a Number: " << endl;

### cin >> input;

### // Convert Input Number into String for validation

### string input\_convert;

### ostringstream convert;

### convert << input;

### input\_convert = convert.str();

### // Digit Validation based on the length of the number.

### unsigned int digits = input\_convert.length();

### if (digits == 1) {

### cout << "Not a valid input, re enter the number" << endl;

### }

### else if (digits == 2) {

### cout << "Number contains " << digits << " digits" << endl;

### i = false;

### }

### else if (digits == 3) {

### cout << "Number contains " << digits << " digits" << endl;

### i = false;

### }

### else if (digits > 3 && digits < 8) {

### cout << "Number contains " << digits << " digits" << endl;

### i = false;

### }

### else if (digits == 8) {

### cout << "Number contains " << digits << " digits" << endl;

### i = false;

### }

### else {

### cout << "Not a valid input, re enter the number" << endl;

### }

### }

### return input;

### }

### int modify(int input){

### // Convert input Number into String for validation

### string input\_string;

### ostringstream convert;

### convert << input;

### input\_string = convert.str();

### //Assign inputLength used to modify the data for transmission according to different sets of rules.

### int inputLength = input\_string.length();

### //Rule set for a two digit number

### if (inputLength == 2) {

### int first\_digit = input / 10;

### int last\_digit = input % 10;

### int swapped\_number = (last\_digit \* 10) + first\_digit;

### return swapped\_number;

### }

### //Rule set for a three digit number

### else if (inputLength == 3) {

### int last\_digit = input % 10;

### int middle\_digit = (input / 10) % 10;

### int first\_digit = input / 100;

### int swapped\_number = (last\_digit \* 100) + (middle\_digit \* 10) + first\_digit;

### return swapped\_number;

### }

### // Rule set for a four digit number.

### else if (inputLength == 4) {

### double array[inputLength];

### array[1] = input % 10;

### array[2] = (input / 10) % 10;

### array[3] = (input / 100) % 10;

### array[4] = input / 1000;

### for (int i = inputLength; i >= 1; i--) {

### array[i] = array[i] \* 3;

### array[i] = array[i] / 10;

### array[i] = fmod(array[i], 1);

### array[i] = array[i] \* 10;

### }

### int product = (array[4] \* 1000) + (array[3] \* 100) + (array[2] \* 10) + array[1] + 1;

### return product;

### }

### //Rule set for a five digit number.

### else if (inputLength == 5) {

### double array[inputLength];

### array[1] = input % 10;

### array[2] = (input / 10) % 10;

### array[3] = (input / 100) % 10;

### array[4] = (input / 1000) % 10;

### array[5] = (input / 10000) % 10;

### for (int i = inputLength; i >= 1; i--) {

### array[i] = array[i] \* 3;

### array[i] = array[i] / 10;

### array[i] = fmod(array[i], 1);

### array[i] = array[i] \* 10;

### }

### int product = (array[5] \* 10000) + (array[4] \* 1000) + (array[3] \* 100) + (array[2] \* 10) + array[1] + 1;

### return product;

### }

### //Rule set for a six digit number.

### else if (inputLength == 6) {

### double array[inputLength];

### array[1] = input % 10;

### array[2] = (input / 10) % 10;

### array[3] = (input / 100) % 10;

### array[4] = (input / 1000) % 10;

### array[5] = (input / 10000) % 10;

### array[6] = (input / 100000) % 10;

### for (int i = inputLength; i >= 1; i--) {

### array[i] = array[i] \* 3;

### array[i] = array[i] / 10;

### array[i] = fmod(array[i], 1);

### array[i] = array[i] \* 10;

### }

### int product = (array[6] \* 100000) + (array[5] \* 10000) + (array[4] \* 1000) + (array[3] \* 100) + (array[2] \* 10) + array[1] + 1;

### return product;

### }

### //Rule set for a seven digit number.

### else if (inputLength == 7) {

### double array[inputLength];

### array[1] = input % 10;

### array[2] = (input / 10) % 10;

### array[3] = (input / 100) % 10;

### array[4] = (input / 1000) % 10;

### array[5] = (input / 10000) % 10;

### array[6] = (input / 100000) % 10;

### array[7] = (input / 1000000);

### for (int i = inputLength; i >= 1; i--) {

### array[i] = array[i] \* 3;

### array[i] = array[i] / 10;

### array[i] = fmod(array[i], 1);

### array[i] = array[i] \* 10;

### }

### int product = (array[7] \* 1000000) + (array[6] \* 100000) + (array[5] \* 10000) + (array[4] \* 1000) + (array[3] \* 100) + (array[2] \* 10) + array[1] + 1;

### return product;

### }

### //Rule set for a eight digit number.

### else if (inputLength == 8) {

### double array[inputLength];

### array[1] = input % 10;

### array[2] = (input / 10) % 10;

### array[3] = (input / 100) % 10;

### array[4] = (input / 1000) % 10;

### array[5] = (input / 10000) % 10;

### array[6] = (input / 100000) % 10;

### array[7] = (input / 1000000) % 10;

### array[8] = (input / 10000000);

### for (int i = inputLength; i >= 1; i--) {

### array[i] = array[i] \* 3;

### array[i] = array[i] / 10;

### array[i] = fmod(array[i], 1);

### array[i] = array[i] \* 10;

### }

### int product = (array[8] \* 10000000) + (array[7] \* 1000000) + (array[6] \* 100000) + (array[5] \* 10000) + (array[4] \* 1000) + (array[3] \* 100) + (array[2] \* 10) + array[1] + 1;

### cout << "Do you want a second level of encryption, (yes or no)?";

### cin >> secondEncryption;

### if (secondEncryption == "yes") {

### int product = (array[1] \* 10000000) + (array[2] \* 1000000) + (array[3] \* 100000) + (array[4] \* 10000) + (array[5] \* 1000) + (array[6] \* 100) + (array[7] \* 10) + array[8] + 1;

### return product;

### }

### else {

### return product;

### }

### }

### }

### int decode(int encrypted) {

### // Convert input Number into String for validation

### string input\_string;

### ostringstream convert;

### convert << encrypted;

### input\_string = convert.str();

### int inputLength = input\_string.length();

### // Decoding Rules set for a 2 digit number

### if (inputLength == 2) {

### int first\_digit = encrypted / 10;

### int second\_digit = encrypted % 10;

### int swapped\_number = (second\_digit \* 10) + first\_digit;

### return swapped\_number;

### }

### // Decoding Rule set for a 3 digit number

### if (inputLength == 3) {

### int last\_digit = encrypted % 10;

### int middle\_digit = (encrypted / 10) % 10;

### int first\_digit = encrypted / 100;

### int swapped\_number = (last\_digit \* 100) + (middle\_digit \* 10) + first\_digit;

### return swapped\_number;

### }

### // Decoding Rule set for a 4 digit number

### if (inputLength == 4) {

### int array[inputLength];

### array[1] = encrypted % 10;

### array[2] = (encrypted / 10) % 10;

### array[3] = (encrypted / 100) % 10;

### array[4] = encrypted / 1000;

### for (int i = inputLength; i >= 1; i--) {

### if (array[i] == 3 || array[i] == 6 || array[i] == 9) {

### array[i] = array[i] / 3;

### }

### else if (array[i] == 2 || array[i] == 5 || array[i] == 8) {

### array[i] = array[i] + 10;

### array[i] = array[i] / 3;

### }

### else if (array[i] == 1 || array[i] == 4 || array[i] == 7) {

### array[i] = array[i] + 20;

### array[i] = array[i] / 3;

### }

### }

### int product = (array[4] \* 1000) + (array[3] \* 100) + (array[2] \* 10) + array[1];

### return product;

### }

### // Decoding Rule set for a 5 digit number

### else if (inputLength == 5) {

### int array[inputLength];

### array[1] = encrypted % 10;

### array[2] = (encrypted / 10) % 10;

### array[3] = (encrypted / 100) % 10;

### array[4] = (encrypted / 1000) % 10;

### array[5] = (encrypted / 10000) % 10;

### for (int i = inputLength; i >= 1; i--) {

### if (array[i] == 3 || array[i] == 6 || array[i] == 9) {

### array[i] = array[i] / 3;

### }

### else if (array[i] == 2 || array[i] == 5 || array[i] == 8) {

### array[i] = array[i] + 10;

### array[i] = array[i] / 3;

### }

### else if (array[i] == 1 || array[i] == 4 || array[i] == 7) {

### array[i] = array[i] + 20;

### array[i] = array[i] / 3;

### }

### }

### int product = (array[5] \* 10000) + (array[4] \* 1000) + (array[3] \* 100) + (array[2] \* 10) + array[1];

### return product;

### }

### // Decoding Rule set for a 6 digit number

### else if (inputLength == 6) {

### int array[inputLength];

### array[1] = encrypted % 10;

### array[2] = (encrypted / 10) % 10;

### array[3] = (encrypted / 100) % 10;

### array[4] = (encrypted / 1000) % 10;

### array[5] = (encrypted / 10000) % 10;

### array[6] = (encrypted / 100000) % 10;

### for (int i = inputLength; i >= 1; i--) {

### if (array[i] == 3 || array[i] == 6 || array[i] == 9) {

### array[i] = array[i] / 3;

### }

### else if (array[i] == 2 || array[i] == 5 || array[i] == 8) {

### array[i] = array[i] + 10;

### array[i] = array[i] / 3;

### }

### else if (array[i] == 1 || array[i] == 4 || array[i] == 7) {

### array[i] = array[i] + 20;

### array[i] = array[i] / 3;

### }

### }

### int product = (array[6] \* 100000) + (array[5] \* 10000) + (array[4] \* 1000) + (array[3] \* 100) + (array[2] \* 10) + array[1];

### return product;

### }

### // Decoding Rule set for a 7 digit number

### else if (inputLength == 7) {

### int array[inputLength];

### array[1] = encrypted % 10;

### array[2] = (encrypted / 10) % 10;

### array[3] = (encrypted / 100) % 10;

### array[4] = (encrypted / 1000) % 10;

### array[5] = (encrypted / 10000) % 10;

### array[6] = (encrypted / 100000) % 10;

### array[7] = (encrypted / 1000000);

### for (int i = inputLength; i >= 1; i--) {

### if (array[i] == 3 || array[i] == 6 || array[i] == 9) {

### array[i] = array[i] / 3;

### }

### else if (array[i] == 2 || array[i] == 5 || array[i] == 8) {

### array[i] = array[i] + 10;

### array[i] = array[i] / 3;

### }

### else if (array[i] == 1 || array[i] == 4 || array[i] == 7) {

### array[i] = array[i] + 20;

### array[i] = array[i] / 3;

### }

### }

### int product = (array[7] \* 1000000) + (array[6] \* 100000) + (array[5] \* 10000) + (array[4] \* 1000) + (array[3] \* 100) + (array[2] \* 10) + array[1];

### return product;

### }

### // Decoding Rule set for a 8 digit number

### else if (inputLength == 8) {

### string secondDecode;

### int array[inputLength];

### array[1] = encrypted % 10;

### array[2] = (encrypted / 10) % 10;

### array[3] = (encrypted / 100) % 10;

### array[4] = (encrypted / 1000) % 10;

### array[5] = (encrypted / 10000) % 10;

### array[6] = (encrypted / 100000) % 10;

### array[7] = (encrypted / 1000000) % 10;

### array[8] = (encrypted / 10000000);

### for (int i = inputLength; i >= 1; i--) {

### if (array[i] == 3 || array[i] == 6 || array[i] == 9) {

### array[i] = array[i] / 3;

### }

### else if (array[i] == 2 || array[i] == 5 || array[i] == 8) {

### array[i] = array[i] + 10;

### array[i] = array[i] / 3;

### }

### else if (array[i] == 1 || array[i] == 4 || array[i] == 7) {

### array[i] = array[i] + 20;

### array[i] = array[i] / 3;

### }

### }

### // If not used a second level of incoding impliment the following decoding rule set

### int product = (array[8] \* 10000000) + (array[7] \* 1000000) + (array[6] \* 100000) + (array[5] \* 10000) + (array[4] \* 1000) + (array[3] \* 100) + (array[2] \* 10) + array[1];

### // If used second level decoding impliment the following decoding rule set

### if (secondEncryption == "yes") {

### int product = (array[1] \* 10000000) + (array[2] \* 1000000) + (array[3] \* 100000) + (array[4] \* 10000) + (array[5] \* 1000) + (array[6] \* 100) + (array[7] \* 10) + array[8];

### return product;

### }

### else {

### return product;

### }

### }

### }

### int main()

### {

### // Title only shown once at start of program

### cout << "Assignment 1: Number Encrypter" << endl;

### //Function that takes user input and outputs into the variable that is passed on to the modification ruleset.

### int input = numberInput();

### // Function that takes user input and encrypts the data based on the amount of digits in the users input

### int encrypted = modify(input);

### // Function that takes the encrypted number and decodes it based on the length of digit

### int decodeNumber = decode(encrypted);

### // Output both the encoded and decoded values as well as a blank line to format the programs output (so the user can read it better)

### cout << "" << endl;

### cout << "Encrypted Number: " << encrypted << endl;

### cout << "Orignal Number: " << decodeNumber << endl;

### return 0;

### }

### Screenshots showing working program (Show all possible outcome):

A screen shot of a computer

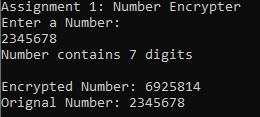
Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screen shot of a computer

Description automatically generated



Screen of a cell phone

Description automatically generated

A screen shot of a social media post

Description automatically generated

## Task 2.11

### Program description

This program uses a formula to calculate the voltage gain based on the frequency and the number of stages in the amplifier.

### Source Code

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

float f, n, v;

float a1, a2, a3;

cout << "The frequency in Hz (f):" << endl;

cin >> f;

cout << "The number of stages in the amplifier (n):" << endl;

cin >> n;

a1 = pow(23, 2.0) + (0.5 \* pow(f, 2.0));

a2 = sqrt(a1);

a3 = 275 / a2;

v = pow(a3, n);

cout << "At a frequency of " << f << " hertz, the voltage gain is " << v << endl;

return 0;

}

### Screenshots showing working program

A screen shot of a social media post

Description automatically generated

## Task 3.8

### Program description

This program produces a text-based table displaying data about a computer.

### Source Code

#include <iostream>

#include <cmath>

#include <iomanip>

using namespace std;

int main()

{

string userName, employeeName, serialNumber, pcType, macAddress;

float cpuSpeed;

unsigned long ipInt;

int noCore;

unsigned short int ipv4P1;

unsigned short int ipv4P2;

unsigned short int ipv4P3;

unsigned short int ipv4P4;

const std::string sep = "|";

cout << "Please enter the user name: ";

cin >> userName;

cout << "Please enter the employee name: ";

cin.ignore();

getline(cin, employeeName, '\n');

cout << "Please enter the PC serial number (max 10 characters): ";

cin >> serialNumber;

cout << "Please enter the PC type (S=Server, D=Desktop, L=Laptop, T=Tablet, P=Phone): ";

cin >> setw(1) >> pcType;

cout << "Please enter the number of cores: ";

cin >> setw(2) >> noCore;

cout << "Please enter the CPU speed (GHz): ";

cin >> cpuSpeed;

cout << "Please enter the MAC Address: ";

cin >> setw(16) >> macAddress;

cout << "Please enter the IPV4 address without the dots (e.g. 111 222 333 444): ";

cin >> ipv4P1 >> ipv4P2 >> ipv4P3 >> ipv4P4;

ipInt = (ipv4P1 \* pow(256., 3)) + (ipv4P2 \* pow(256., 2)) + (ipv4P3 \* 256) + (ipv4P4);

// Output Table

cout << "+" << setfill('-') << setw(71) << "+" << endl;

cout << setfill(' ') << left << sep << setw(8) << "User: " << userName << sep << setw(16) << "Employee: " << employeeName << sep << setw(8) << "Serial: " << serialNumber << sep << endl;

cout << setfill(' ') << left << sep << setw(8) << "Type: " << pcType << sep << setw(7) << "Cores: " << noCore << sep << setw(5) << "Speed: " << cpuSpeed << "GHz" << sep << setw(5) << "MAC: " << macAddress << sep << setw(6) << "IP: " << ipv4P1 << "." << ipv4P2 << "." << ipv4P3 << "." << ipv4P4 << sep << endl;

cout << setfill(' ') << left << setw(20) << sep << "IP Address as Integer: " << ipInt << right << setw(19) << sep << endl;

cout << "+" << setfill('-') << setw(71) << "+" << endl;

// end of Output Table

return 0;

}

### Screenshots showing working program

A picture containing black, text, photo

Description automatically generated

## Task 4.9

### Program description

This program tells the user what they should do on a day based on whether it’s a weekday or week end and weather or not its raining outside.

### Source Code

#include <iostream>

using namespace std;

int main() {

int raining, a, x;

bool i = true;

while(i) {

cout << "Please enter the day of the week as a number (Monday = 1, Sunday = 7)" << endl;

cin >> a;

if (a < 6) {

cout << "Is it raining outside? (1 for yes, 0 for no)" << endl;

cin >> raining;

if (raining) {

cout << "Take an umbrella!" << endl;

}

else {

cout << "Go to your university class!" << endl;

}

}

else {

cout << "Is it raining outside? (1 for yes, 0 for no)" << endl;

cin >> raining;

if (raining) {

cout << "Read in bed!"<< endl;

}

else {

cout << "Go out and have fun!" << endl;

}

}

cout << "Do you want to continue? (1 for yes, 0 for no)" << endl;

cin >> x;

if (!x) {

i = false;

cout << "Exiting program!" << endl;

}

}

return 0;

}

### Screenshots showing working program

