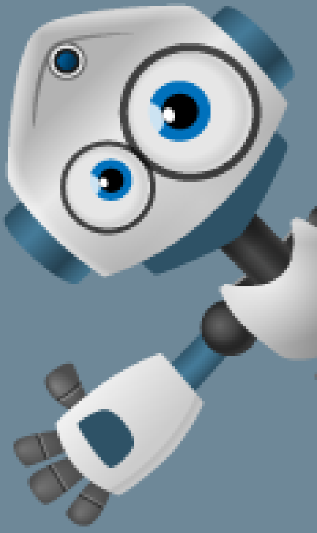
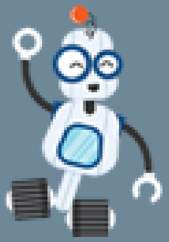
RSA Implantation By C++

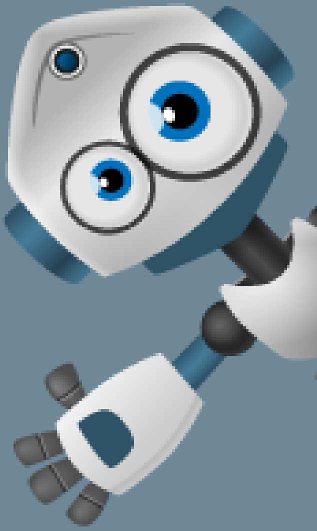
Team Members:



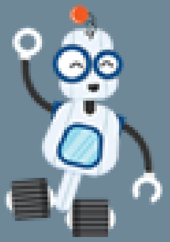
Ibrahim Hamada Ibrahim	201800739
Ayatullah AbdElRahman	201801887
Seif Mohamed Fikry	201801282
Abdullah Kamal Muhammad	201801271



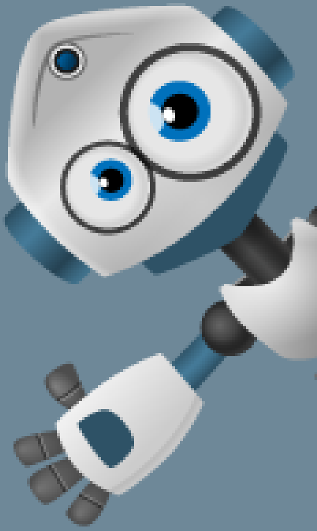
Outline:



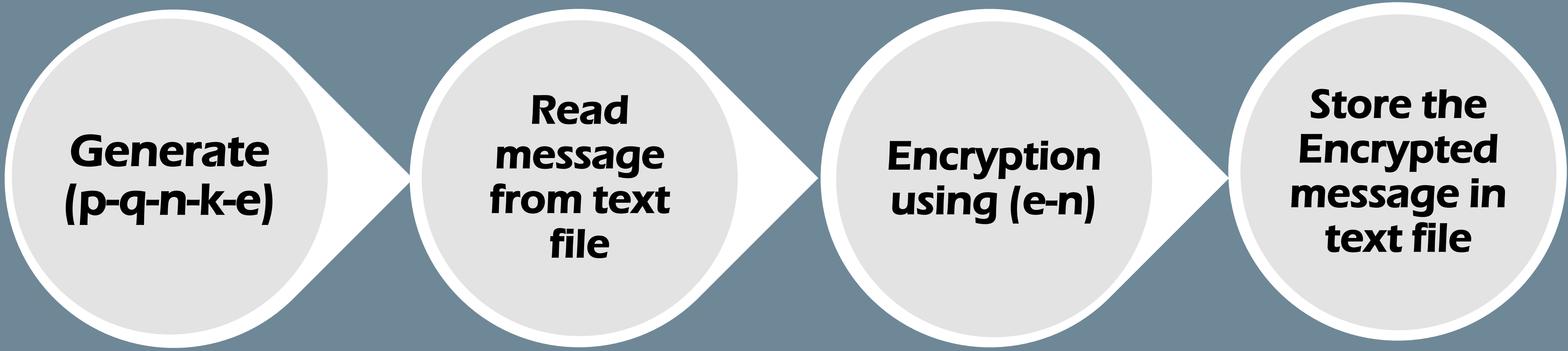
1) Introduction
2) isPrime , GeneratePrime , PublicKey Functions
3) GCD , Coprime Functions
4) MultiplicativeInverse , PrivateKey Functions
5) Modular Exponentiation , ReadOriginalMessage Functions
6) ReadEncryptedMessage , Encryption , Decryption, Store Functions



Introduction:



Encryption:

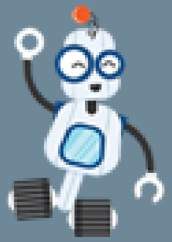


Generate (p-q-n-k-e)

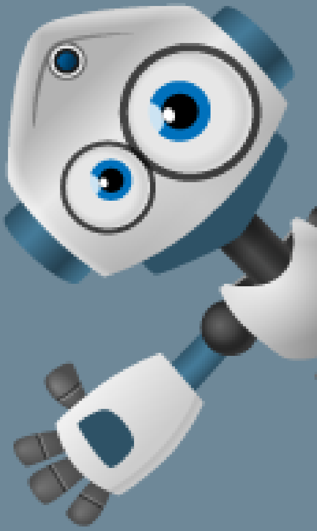
Read message from text file

Encryption using (e-n)

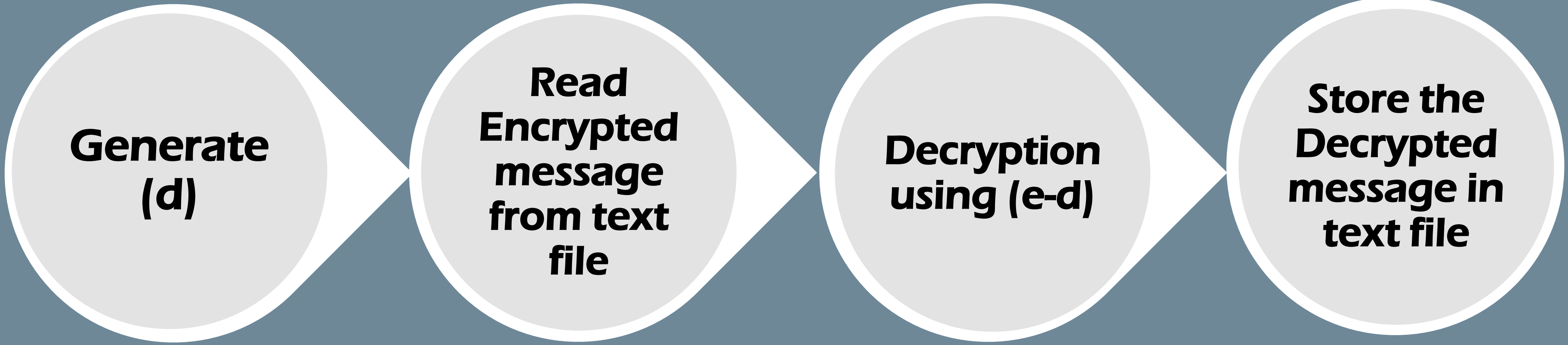
Store the Encrypted message in text file



Introduction:



Decryption:

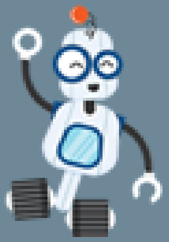


Generate (d)

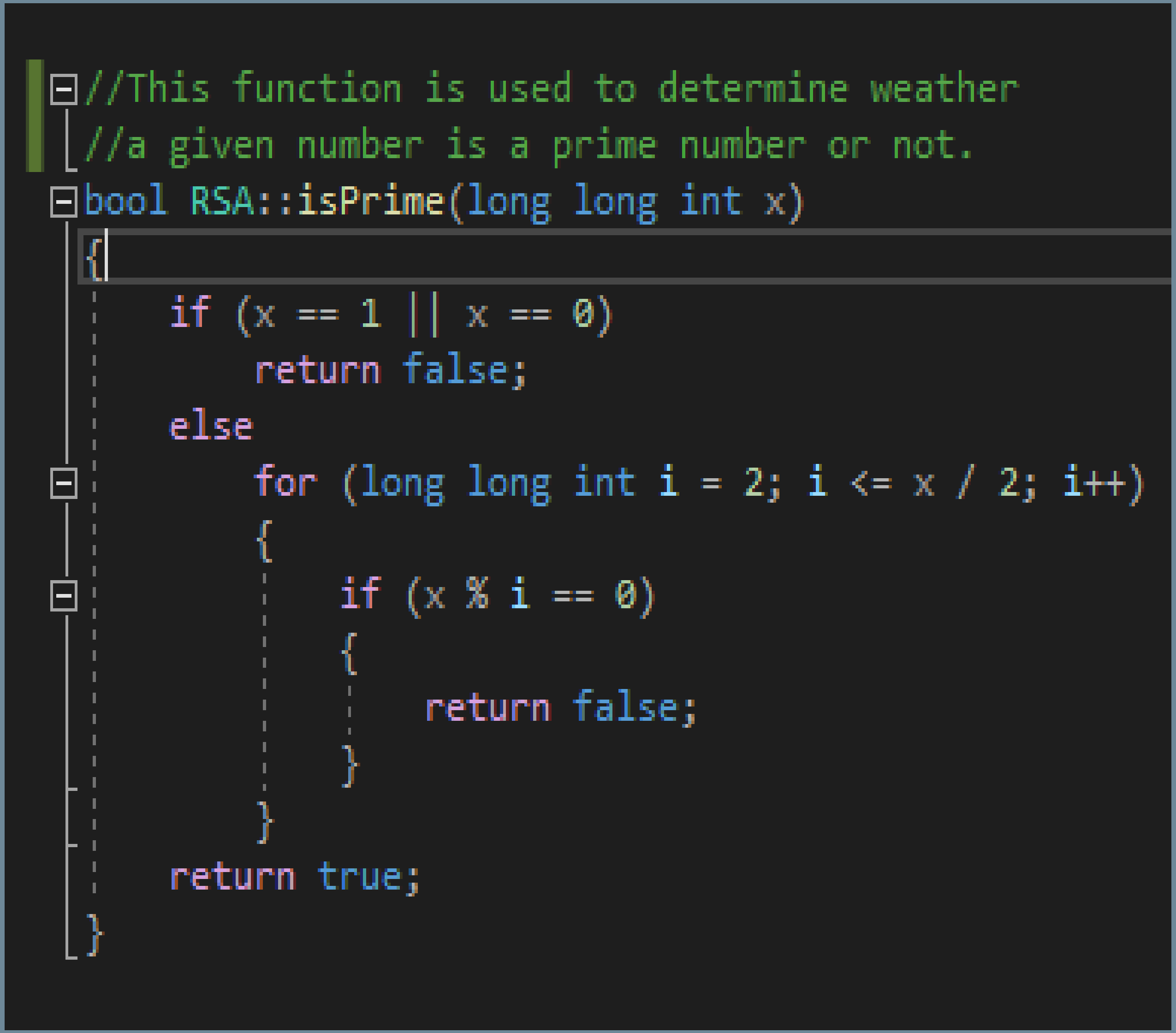
Read Encrypted message from text file

Decryption using (e-d)

Store the Decrypted message in text file



isPrime Function:



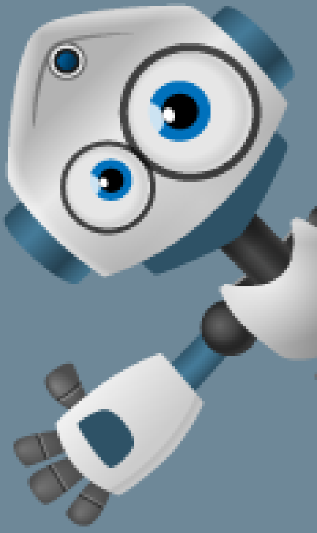
```
//This function is used to determine weather
//a given number is a prime number or not.
bool RSA::isPrime(long long int x)
{
    if (x == 1 || x == 0)
        return false;
    else
        for (long long int i = 2; i <= x / 2; i++)
        {
            if (x % i == 0)
            {
                return false;
            }
        }
    return true;
}
```

-Algorithm:

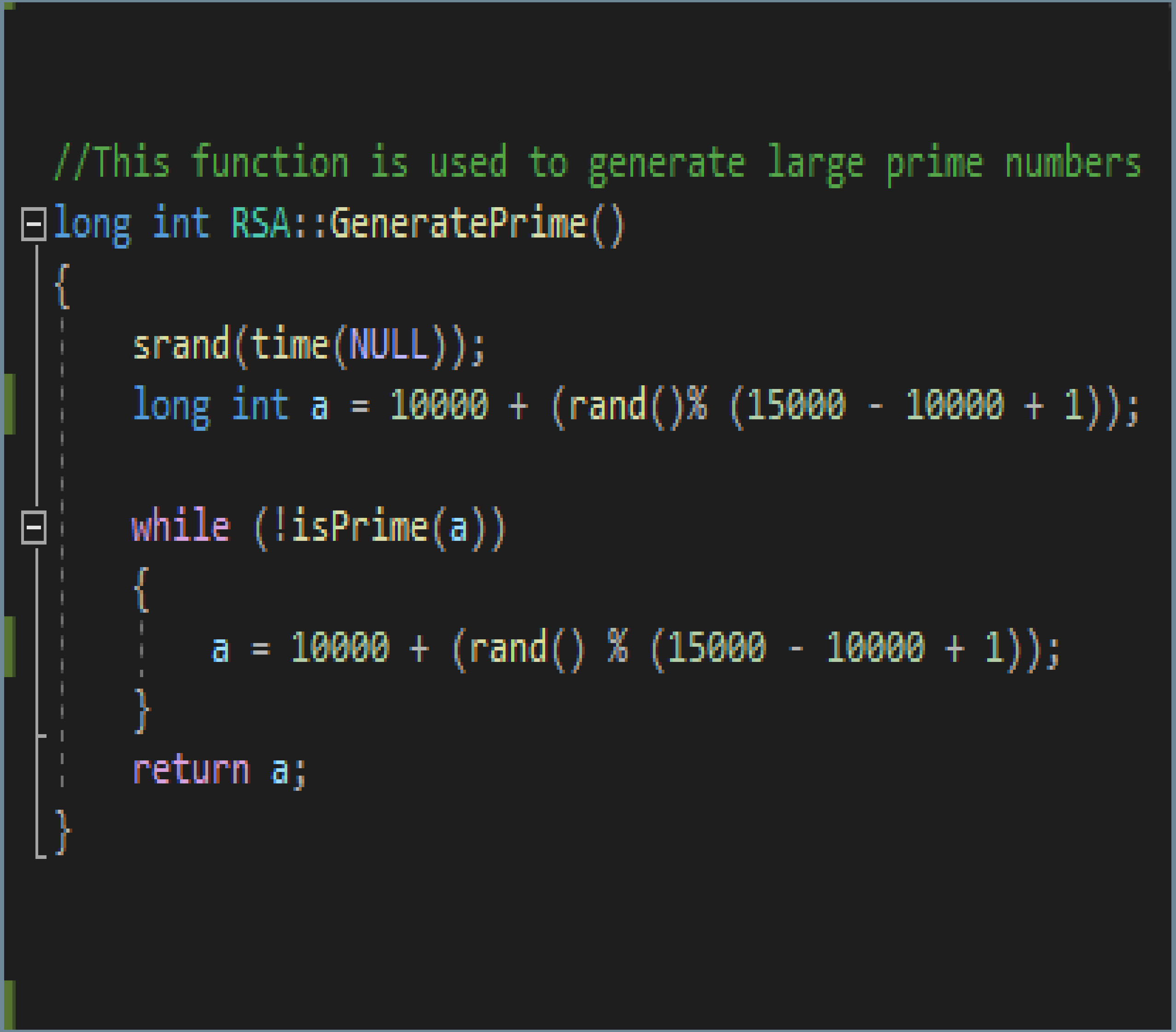
The Boolean function checks whether “x” is primer or not by:

1- checking whether the number equals 1 or 0.

2- checking whether the number is divisible by any integer smaller than it



GeneratePrime Function:



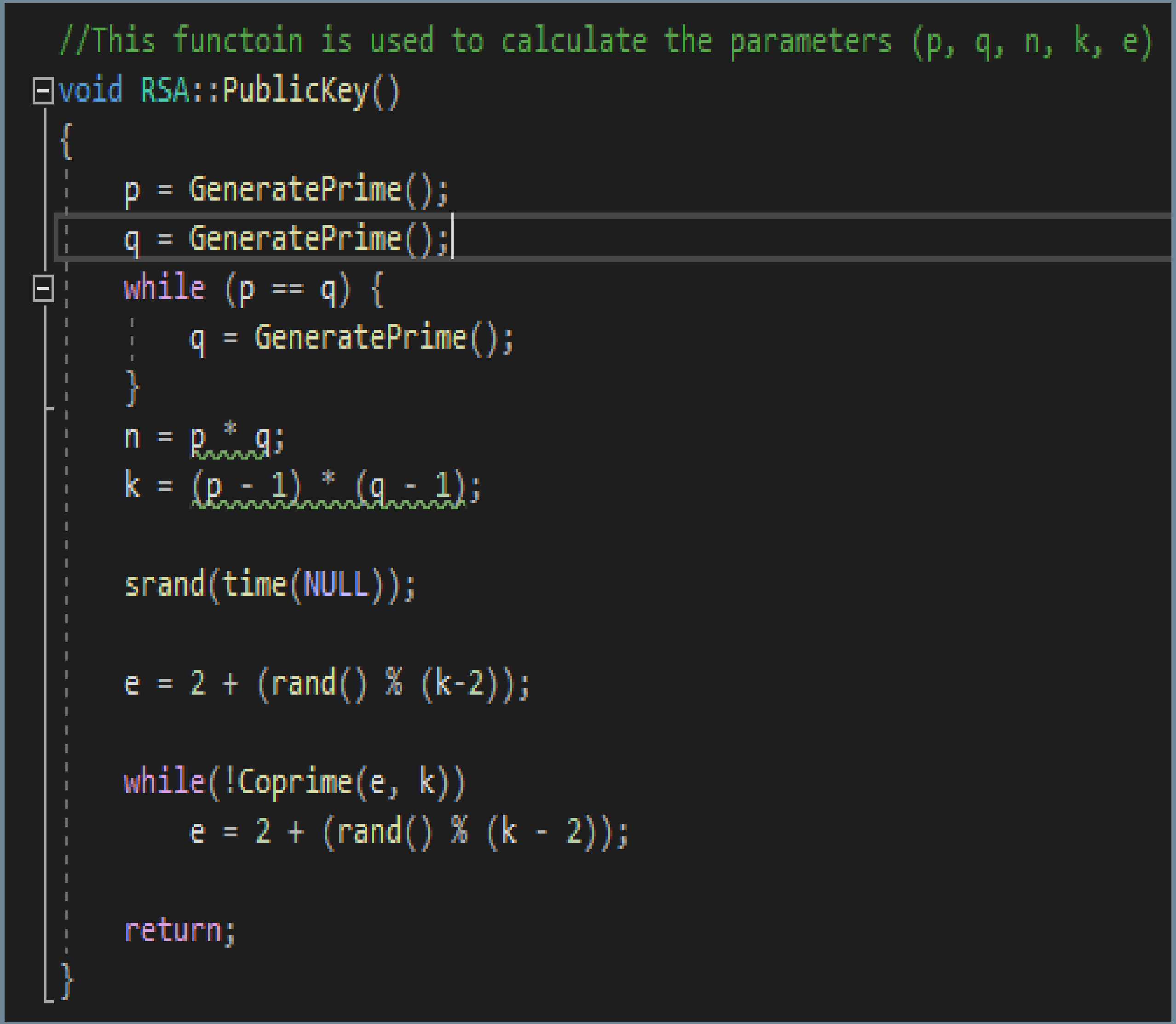
```
//This function is used to generate large prime numbers
long int RSA::GeneratePrime()
{
    srand(time(NULL));
    long int a = 10000 + (rand()% (15000 - 10000 + 1));

    while (!isPrime(a))
    {
        a = 10000 + (rand() % (15000 - 10000 + 1));
    }
    return a;
}
```

-Algorithm:

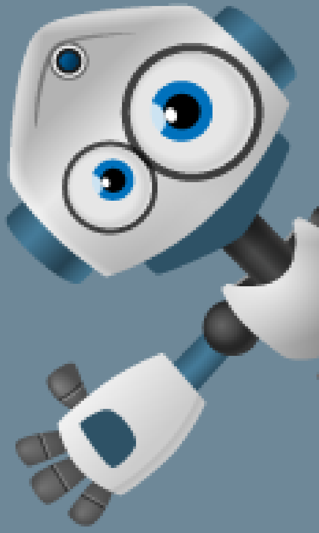
1- The function generates a random number between 10000 and 15000

2- the generated number is checked whether it is prime or not by calling (**isprime**) function and passing the random number as an argument.

PublicKey Function:



```
//This functoin is used to calculate the parameters (p, q, n, k, e)
void RSA::PublicKey()
{
    p = GeneratePrime();
    q = GeneratePrime();
    while (p == q) {
        q = GeneratePrime();
    }
    n = p * q;
    k = (p - 1) * (q - 1);

    srand(time(NULL));

    e = 2 + (rand() % (k-2));

    while(!Coprime(e, k))
        e = 2 + (rand() % (k - 2));

    return;
}
```



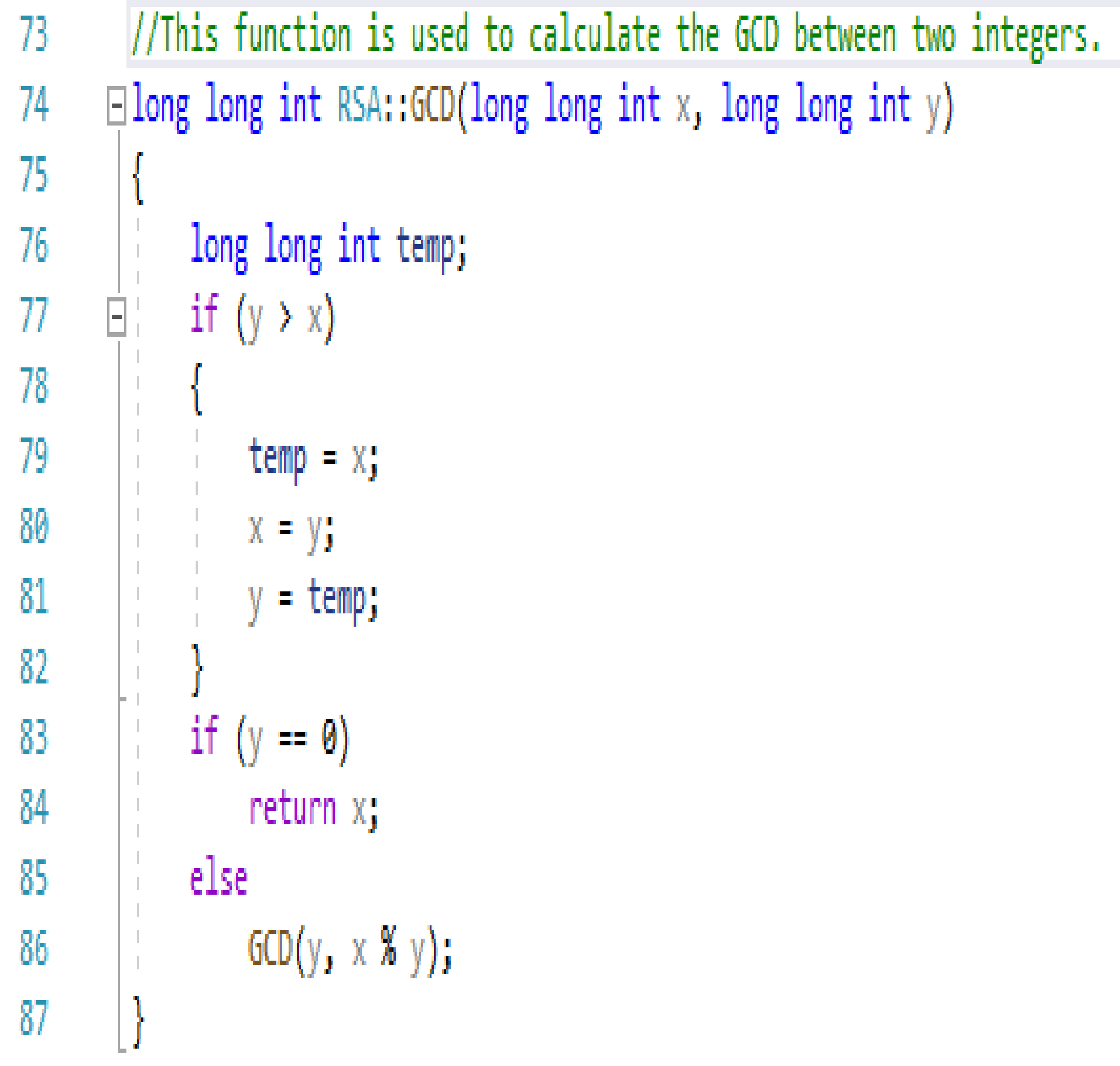
-Algorithm:

1- The function generates 2 different prime numbers.

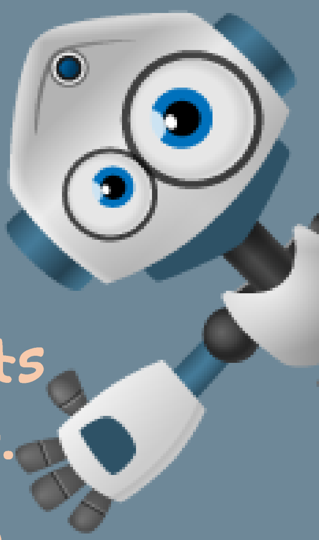
2- The modulus of encrypting and decrypting (n) is calculated.

3- The encryption key is calculated.

GCD Function:



```
//This function is used to calculate the GCD between two integers.
long long int RSA::GCD(long long int x, long long int y)
{
    long long int temp;
    if (y > x)
    {
        temp = x;
        x = y;
        y = temp;
    }
    if (y == 0)
        return x;
    else
        GCD(y, x % y);
}
```



-Algorithm:

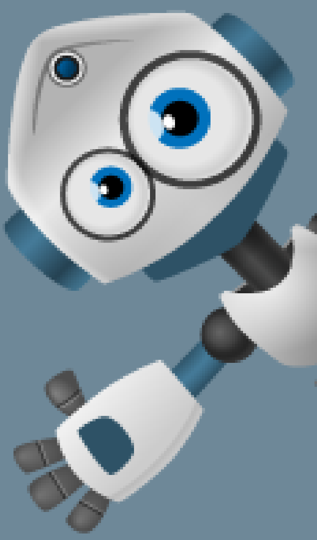
1- This function takes 2 arguments that we want to know their GCD.

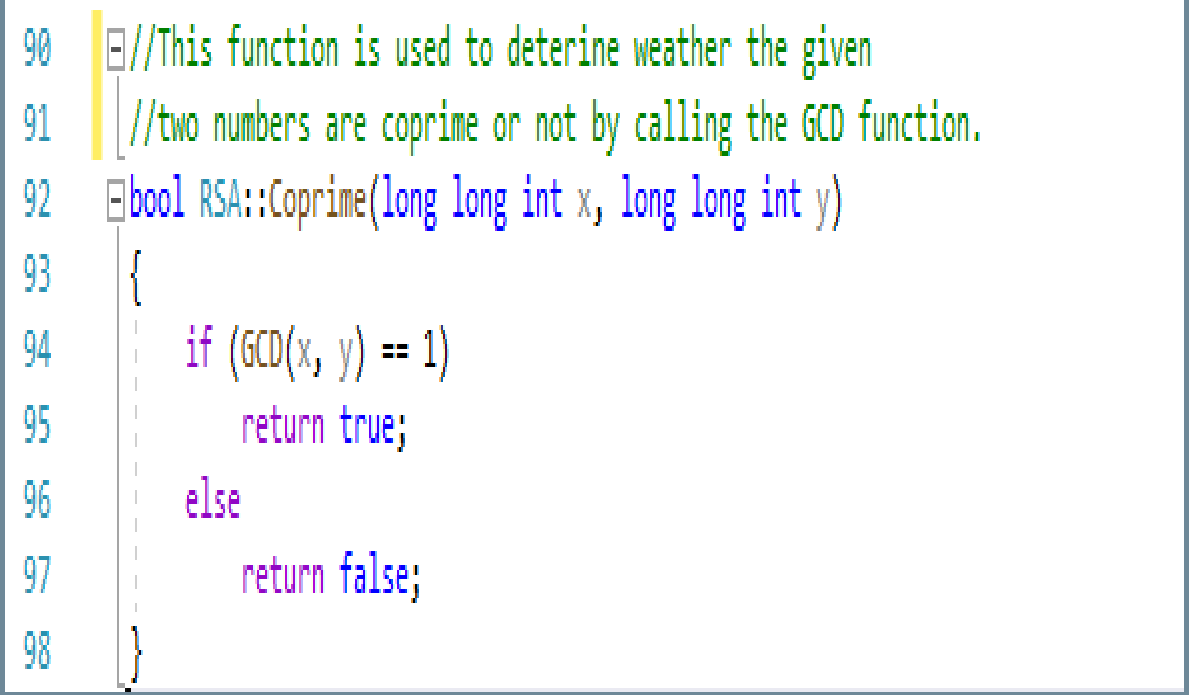
2-it checks the greatest of them then settle at as the first one.

3-if the divisor is zero then the GCD will be the dividend itself .

4- if not, then using the recursion method it will check the GCD between the dividend and the remainder till the remainder reaches zero, the GCD will be the previous remainder.

Coprime Function:

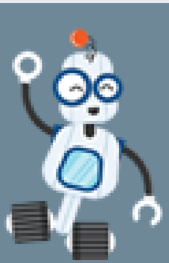




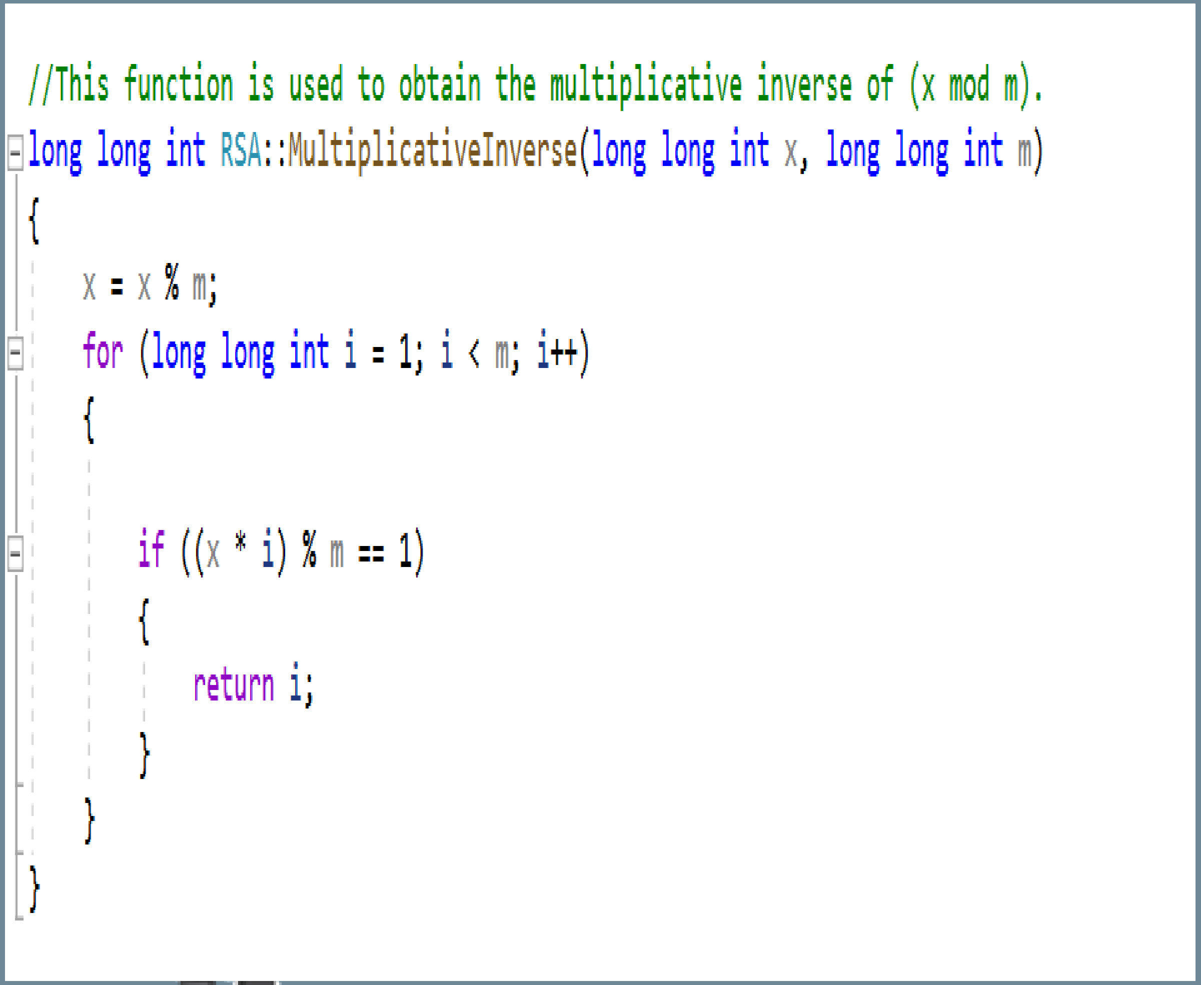
```
//This function is used to deterine weather the given
//two numbers are coprime or not by calling the GCD function.
bool RSA::Coprime(long long int x, long long int y)
{
    if (GCD(x, y) == 1)
        return true;
    else
        return false;
}
```

-Algorithm:

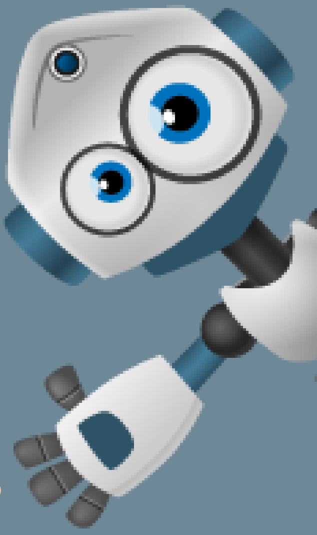
this bool function checks if the GCD was ‘one’ , then those numbers are coprime, so the return is true else then they were not coprime so return is false.



MultiplicativeInverse Function:



```
//This function is used to obtain the multiplicative inverse of (x mod m).
long long int RSA::MultiplicativeInverse(long long int x, long long int m)
{
    x = x % m;
    for (long long int i = 1; i < m; i++)
    {

        if ((x * i) % m == 1)
        {
            return i;
        }
    }
}
```

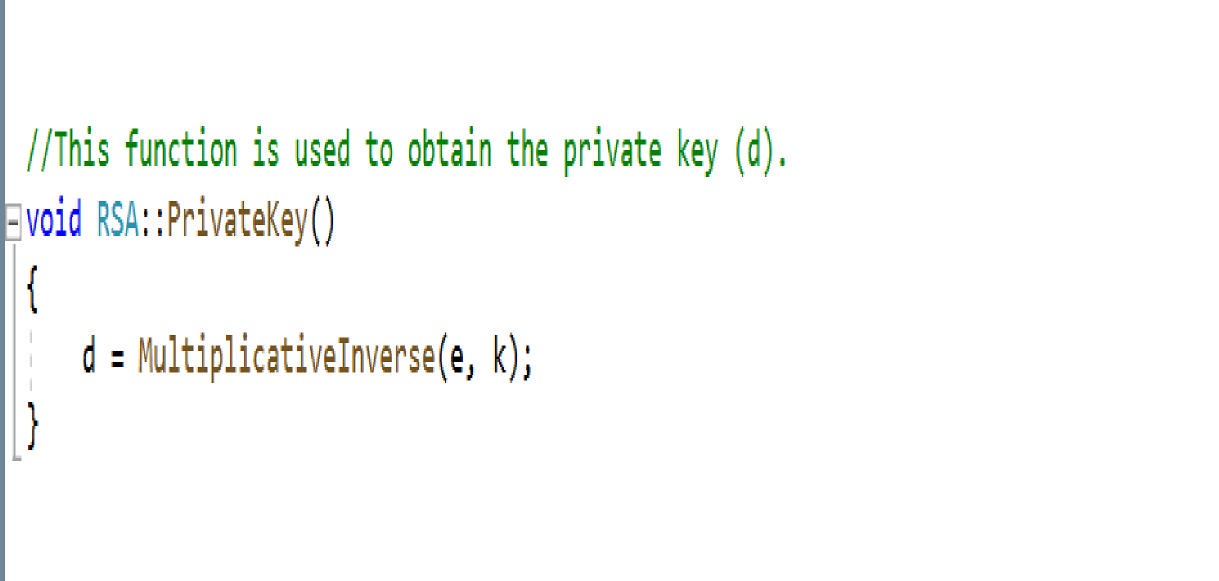


-Algorithm:

1- calculating the remainder of dividing x by m to reduce its value regarding to m .

2- By using a for loop to calculate the multiplicative inverse that makes the remainder of its multiplication with x to m equals to 1 .

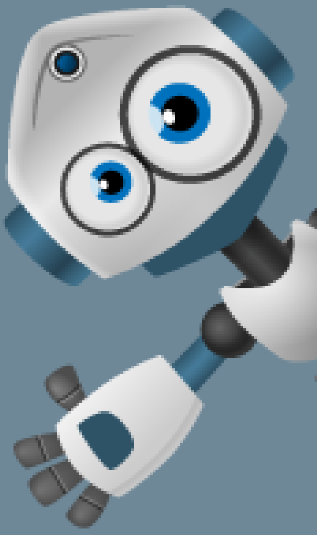
PrivateKey Function:

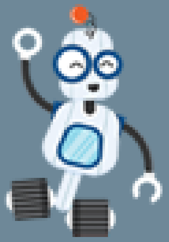


```
//This function is used to obtain the private key (d).
void RSA::PrivateKey()
{
    d = MultiplicativeInverse(e, k);
}
```

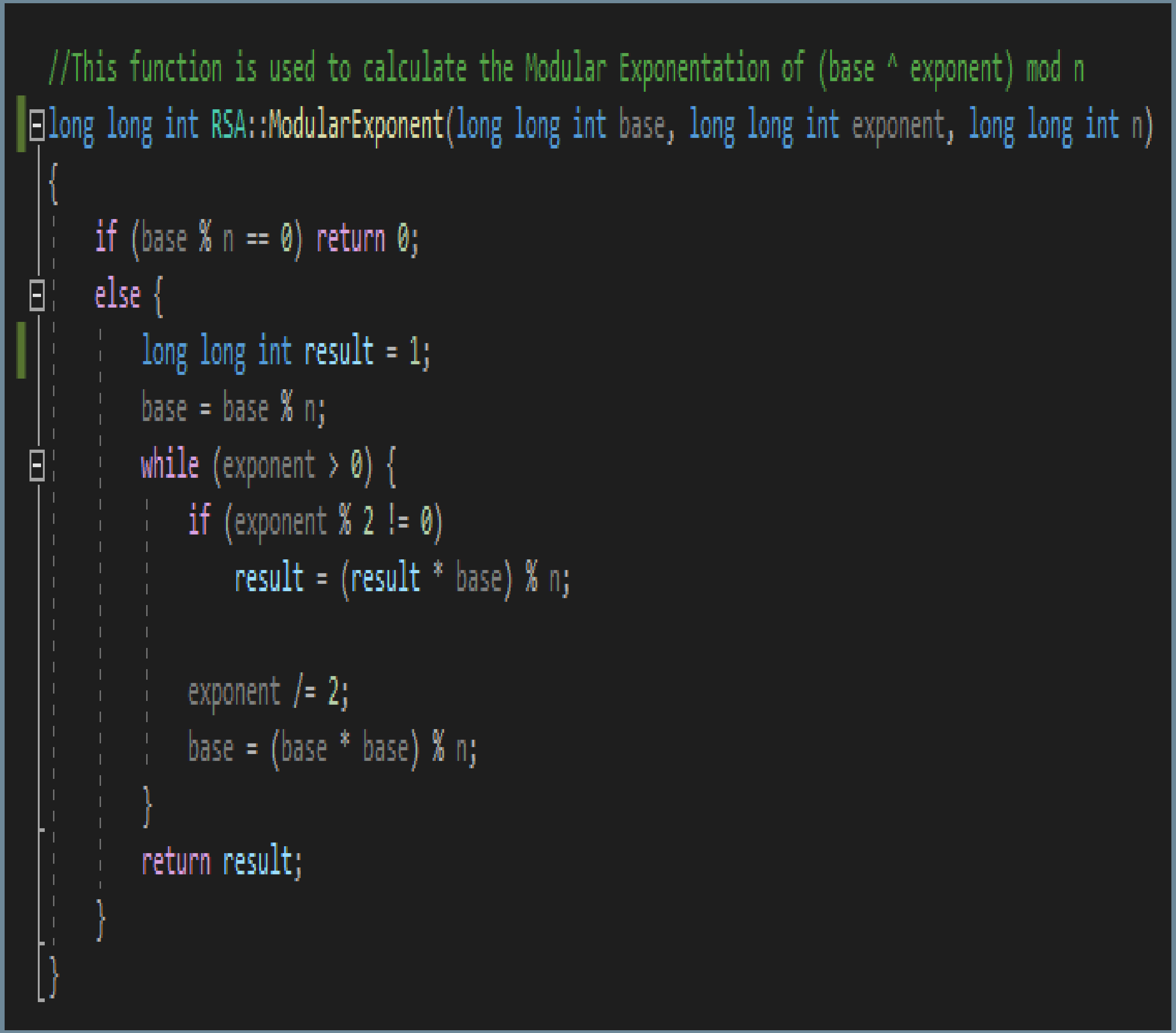
-Algorithm:

From the formula “d*e=1 (mod k)”,it uses the function of multiplicative inverse to determine “d” the private key.



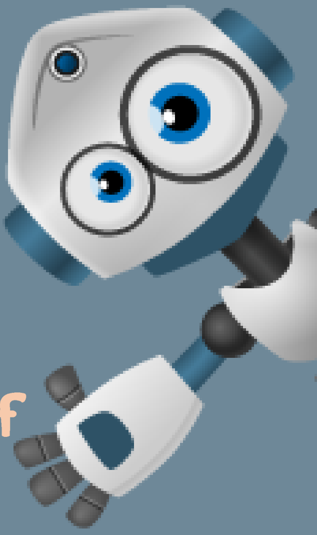


ModularExponent Function:



```
//This function is used to calculate the Modular Exponentation of (base ^ exponent) mod n
long long int RSA::ModularExponent(long long int base, long long int exponent, long long int n)
{
    if (base % n == 0) return 0;
    else {
        long long int result = 1;
        base = base % n;
        while (exponent > 0) {
            if (exponent % 2 != 0)
                result = (result * base) % n;

            exponent /= 2;
            base = (base * base) % n;
        }
        return result;
    }
}
```

-Algorithm:



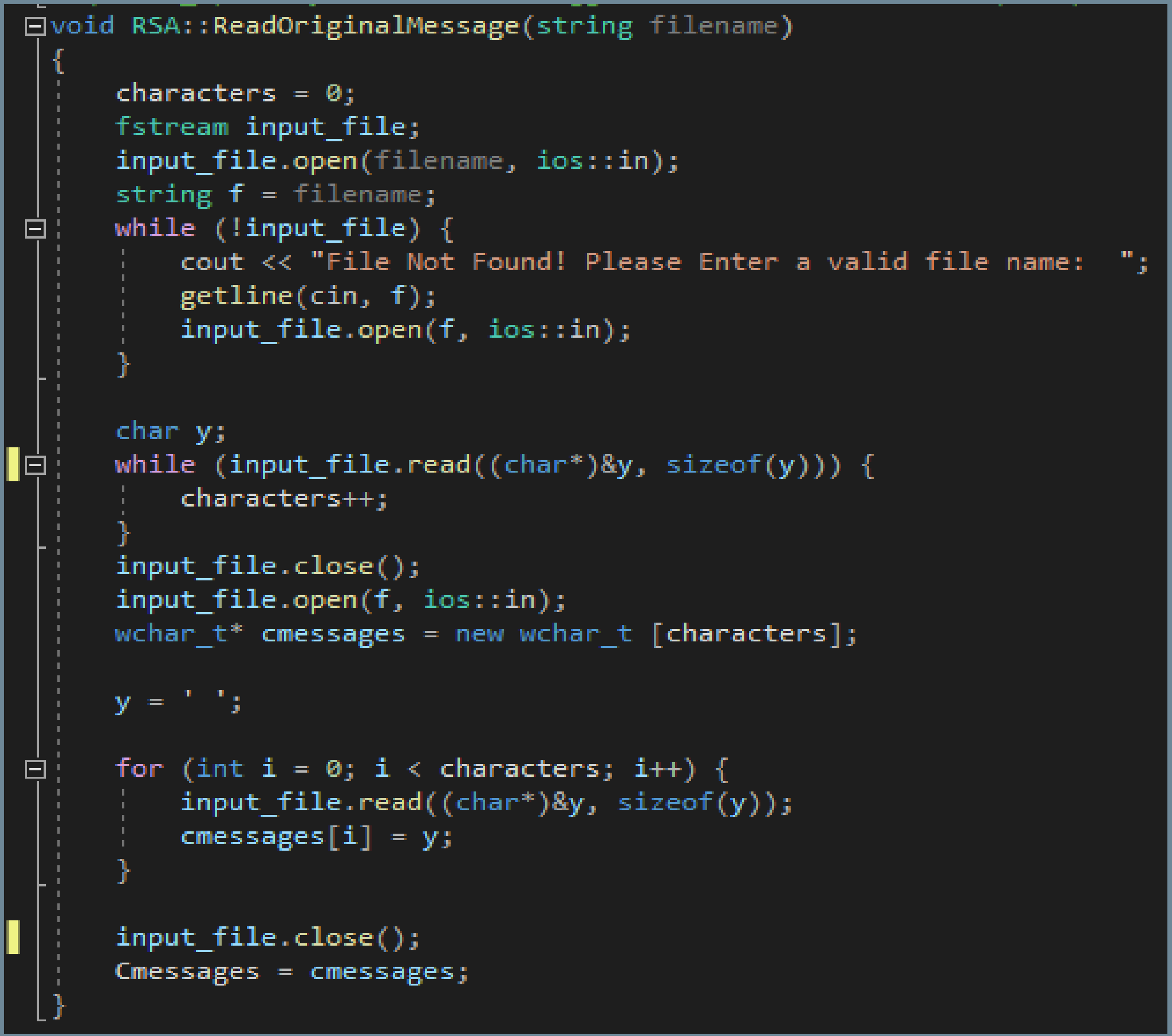
1- Check if (base mod n) is 0, and if it is 0, the function will return.

2- If the (base mod n != 0), a while loop will be made until the (exponent = 0).

3- In every iteration, the (exponent) will be divided by 2 and the base will be multiplied by itself.

4- If (exponent mod 2 != 0), the result will be multiplied by base all under mod n.

ReadOriginalMessage Function:

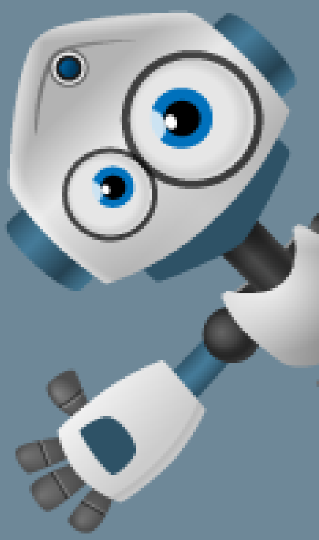


```
void RSA::ReadOriginalMessage(string filename)
{
    characters = 0;
    fstream input_file;
    input_file.open(filename, ios::in);
    string f = filename;
    while (!input_file) {
        cout << "File Not Found! Please Enter a valid file name:  ";
        getline(cin, f);
        input_file.open(f, ios::in);
    }

    char y;
    while (input_file.read((char*)&y, sizeof(y))) {
        characters++;
    }
    input_file.close();
    input_file.open(f, ios::in);
    wchar_t* cmessages = new wchar_t [characters];

    y = ' ';

    for (int i = 0; i < characters; i++) {
        input_file.read((char*)&y, sizeof(y));
        cmessages[i] = y;
    }

    input_file.close();
    Cmessages = cmessages;
}
```

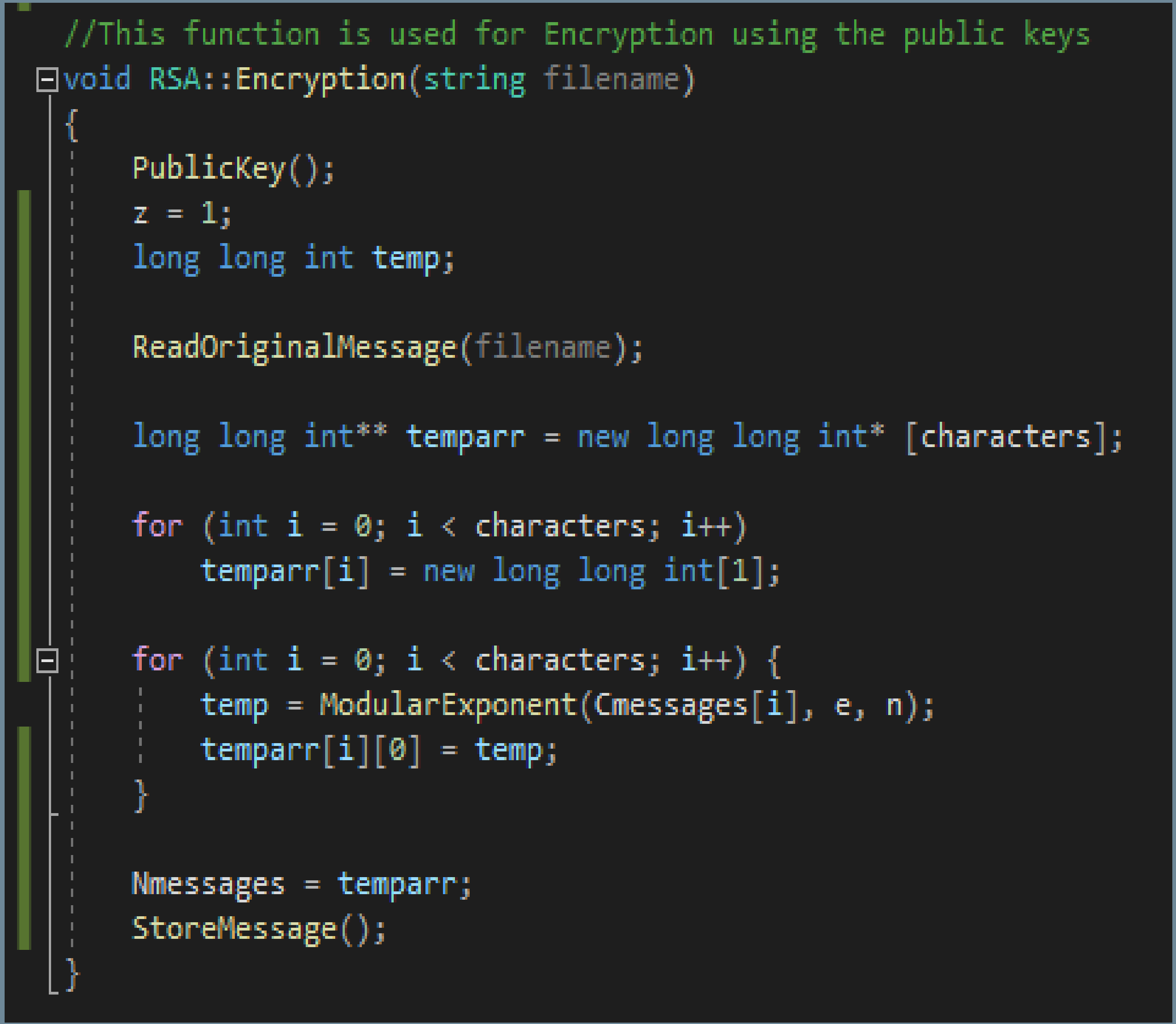
-Algorithm:

1- This function takes the (text file) of the original message.

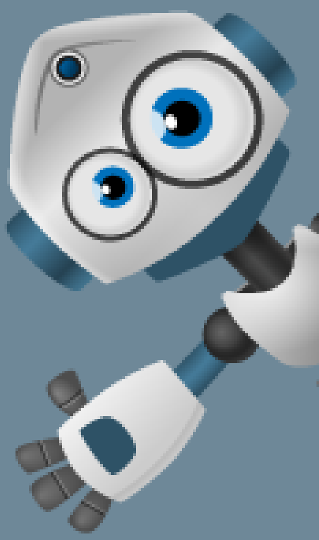
2- Get the number of character and Define a (wide char array) (cmessages) in which the message' s characters will be stored.



Encryption Function:



```
//This function is used for Encryption using the public keys
void RSA::Encryption(string filename)
{
    PublicKey();
    z = 1;
    long long int temp;

    ReadOriginalMessage(filename);

    long long int** temparr = new long long int* [characters];

    for (int i = 0; i < characters; i++)
        temparr[i] = new long long int[1];

    for (int i = 0; i < characters; i++) {
        temp = ModularExponent(Cmessages[i], e, n);
        temparr[i][0] = temp;
    }

    Nmessages = temparr;
    StoreMessage();
}
```



-Algorithm:

1- Generate the (Public Keys).

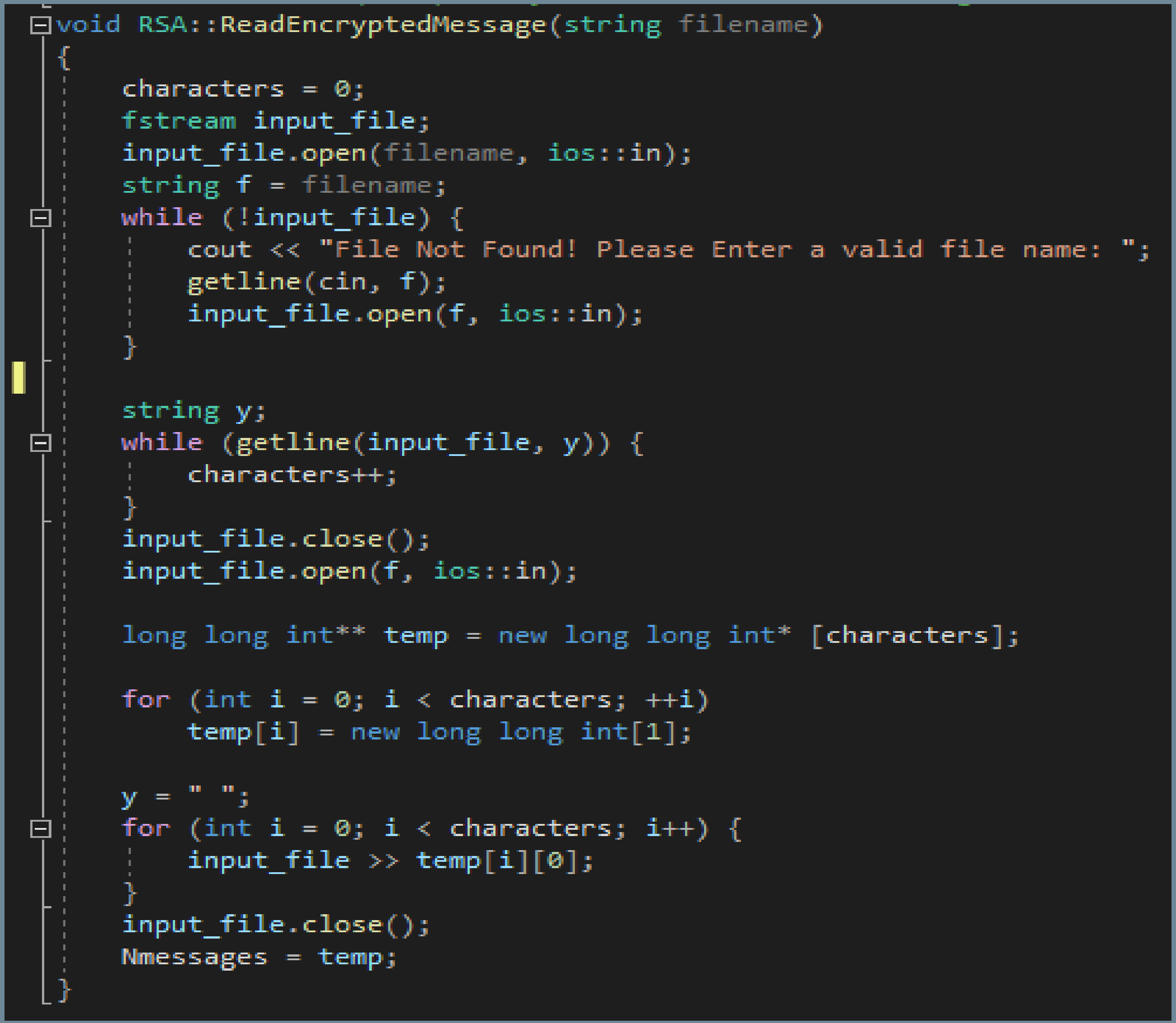
2- Read Original Message.

3- Define a (long long int 2D array) in which the ASCII Code of characters of the Encrypted Message will be stored.

4- The Original Message is Encrypted using ModularExponent Function.

5- The Encrypted Message is stored in the (int 2D array), and then stored in a (text file).

ReadEncryptedMessage Function:



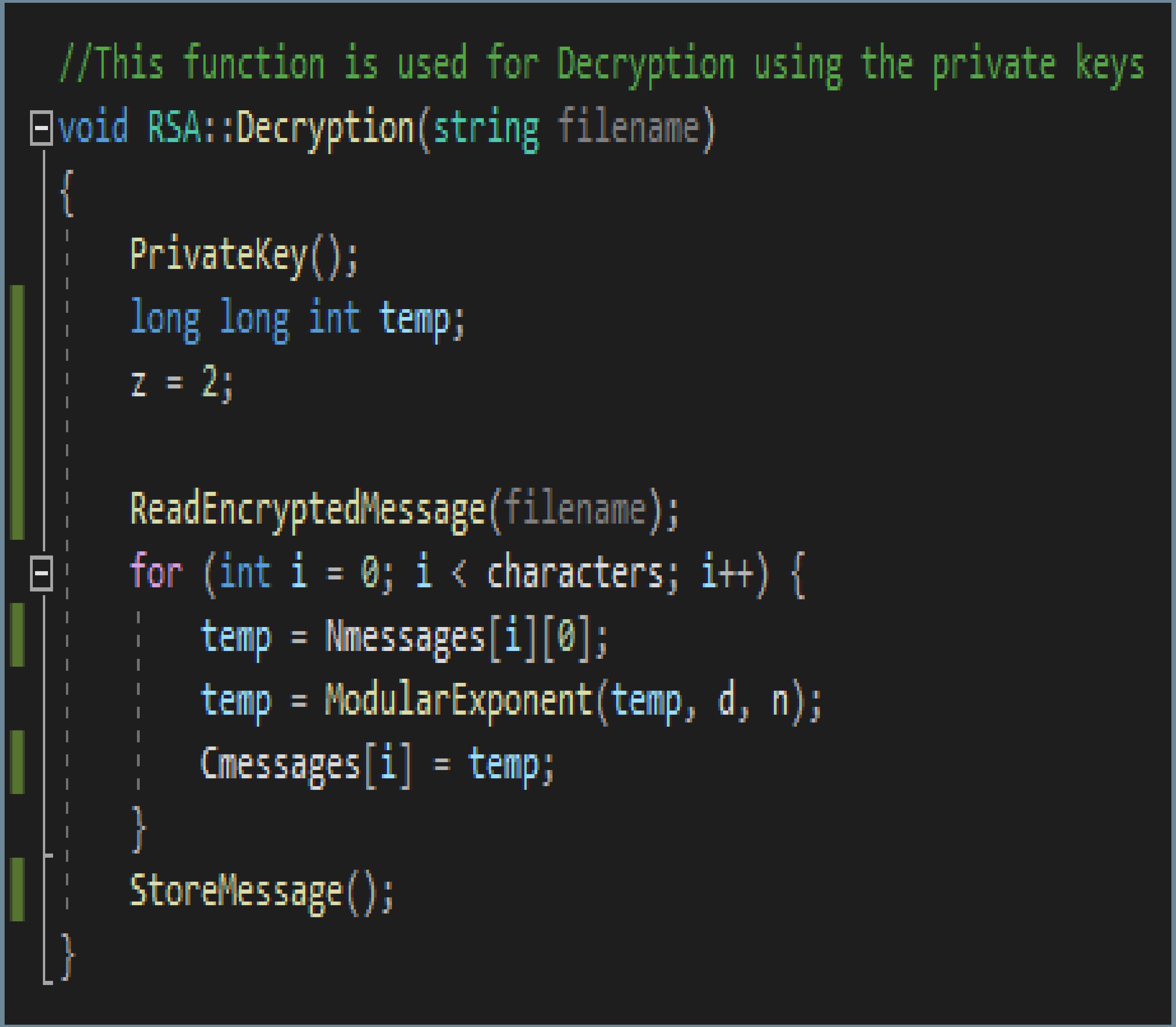
```
void RSA::ReadEncryptedMessage(string filename)
{
    characters = 0;
    fstream input_file;
    input_file.open(filename, ios::in);
    string f = filename;
    while (!input_file) {
        cout << "File Not Found! Please Enter a valid file name: ";
        getline(cin, f);
        input_file.open(f, ios::in);
    }

    string y;
    while (getline(input_file, y)) {
        characters++;
    }
    input_file.close();
    input_file.open(f, ios::in);

    long long int** temp = new long long int* [characters];

    for (int i = 0; i < characters; ++i)
        temp[i] = new long long int[1];

    y = " ";
    for (int i = 0; i < characters; i++) {
        input_file >> temp[i][0];
    }
    input_file.close();
    Nmessages = temp;
}
```

-Algorithm:

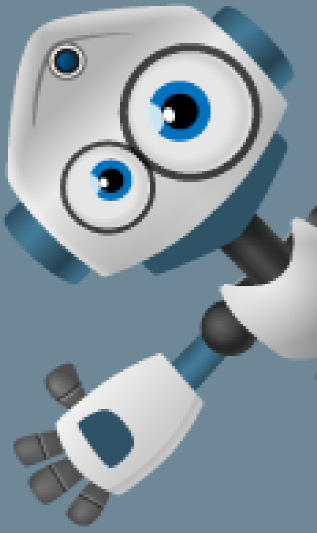
1- This function takes the (text file) of the Encrypted message.

2- Get the number of character and Define a (long long int 2D array) in which the Encrypted message' s characters will be stored.

Decryption Function:



```
//This function is used for Decryption using the private keys
void RSA::Decryption(string filename)
{
    PrivateKey();
    long long int temp;
    z = 2;

    ReadEncryptedMessage(filename);
    for (int i = 0; i < characters; i++) {
        temp = Nmessages[i][0];
        temp = ModularExponent(temp, d, n);
        Cmessages[i] = temp;
    }
    StoreMessage();
}
```



-Algorithm:

1- Generate the (Private Keys).

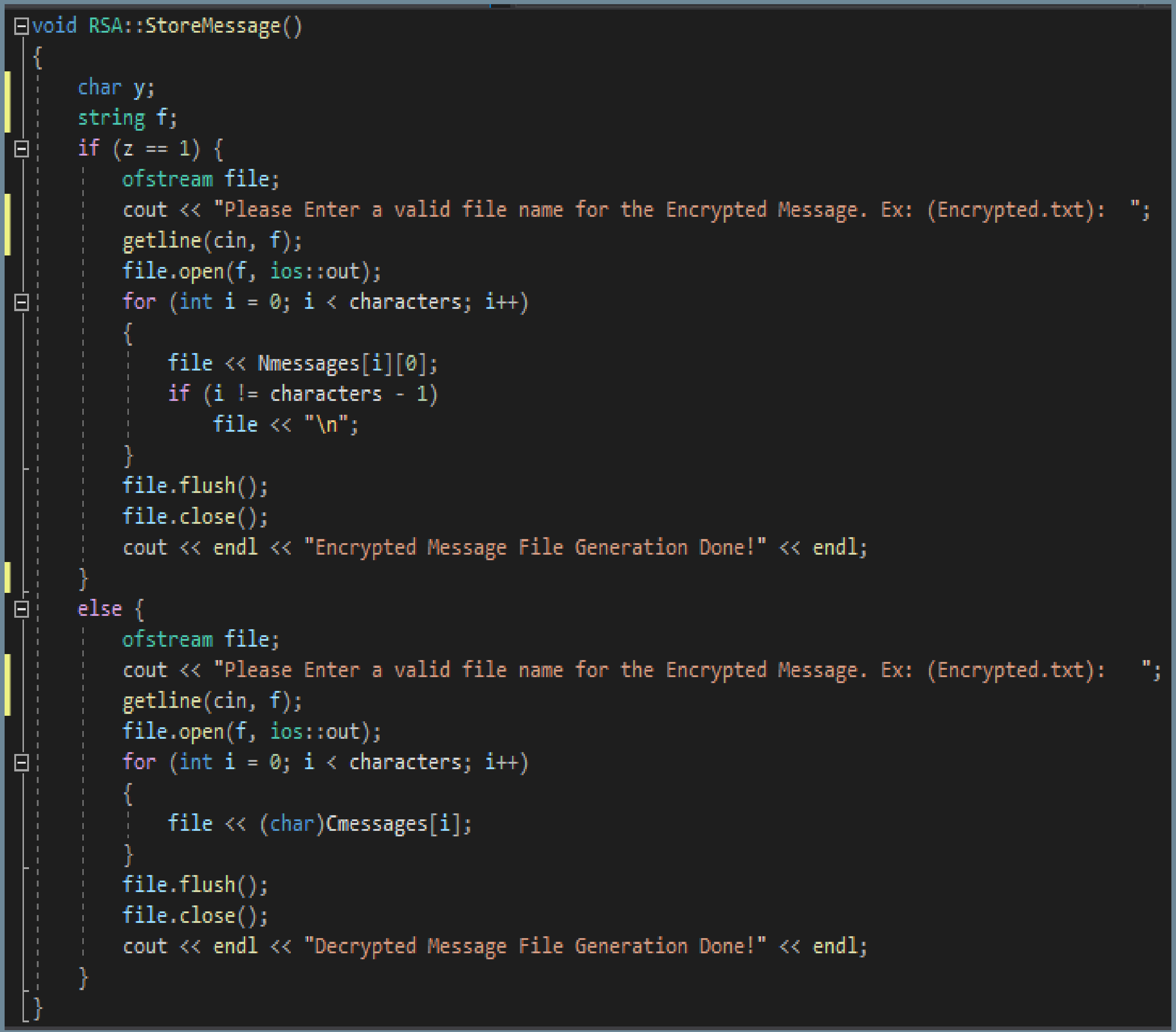
2- Read Encrypted Message.

3- Define a (wide char array) in which the ASCII Code of characters of the Decrypted Message will be stored.

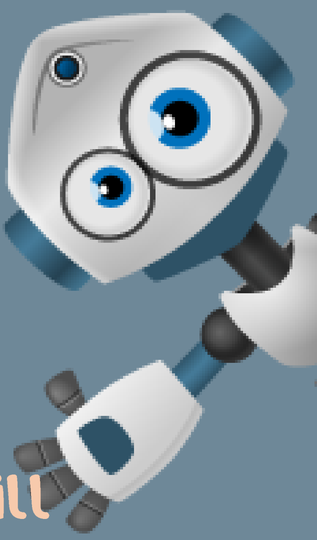
4- The Encrypted Message is Decrypted using ModularExponent Function.

5- The Decrypted Message is stored in the (wide char array), and then stored in a (text file).

StoreMessage Function:



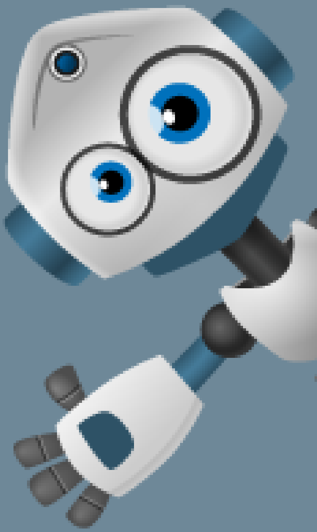
```cpp
void RSA::StoreMessage()
{
    char y;
    string f;
    if (z == 1) {
        ofstream file;
        cout << "Please Enter a valid file name for the Encrypted Message. Ex: (Encrypted.txt):  ";
        getline(cin, f);
        file.open(f, ios::out);
        for (int i = 0; i < characters; i++)
        {
            file << Nmessages[i][0];
            if (i != characters - 1)
                file << "\n";
        }
        file.flush();
        file.close();
        cout << endl << "Encrypted Message File Generation Done!" << endl;
    }
    else {
        ofstream file;
        cout << "Please Enter a valid file name for the Encrypted Message. Ex: (Encrypted.txt):   ";
        getline(cin, f);
        file.open(f, ios::out);
        for (int i = 0; i < characters; i++)
        {
            file << (char)Cmessages[i];
        }
        file.flush();
        file.close();
        cout << endl << "Decrypted Message File Generation Done!" << endl;
    }
}
```



-Algorithm:

1- If it is an Encrypted Message, the elements of (int 2D array) will be stored in a (text file) line by line.

2- If it is an Decrypted Message, the elements of (Char array) will be stored in a (text file) Character by Character.



THE END