// FactorNode2.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include "iostream"

#include "FactorClass.h"

#include "LongNumber.h"

using namespace std;

// Create 1 and 0 as Long Numbers - here so they have global scope and don't need to reinitialise in each routine

LongNumber One(1);

LongNumber Zero(1);

int main(int argc, char\* argv[])

{

// check passed a number to factor

if (argc <= 1)

{

cout << "Insufficient arguments" << endl;

return 0;

}

// Get the number to be factorised from the command line and create as long number

char \* Target = argv[1];

LongNumber LNTarget(Target);

int iTarget = LNTarget[0];

cout << "Factoriser START\n";

cout << "Input Number : ";

PrintLongNumberLR(LNTarget);

// Set One as a Long Number

One.SetLongNumber(1, 0);

One.SetLongNumberBase(10);

One.SetLongNumberSign(1);

// Create zero as a Long Number

Zero.SetLongNumber(0, 0);

Zero.SetLongNumberBase(10);

Zero.SetLongNumberSign(1);

// create head of list

FactorNode FHead(nullptr, 0, Zero, Zero, Zero, Zero, false);

FactorNode \* FHeadPt = & FHead;

LongNumber CountA;

LongNumber CountB;

LongNumber Mult;

LongNumber Remainder;

// Create the first level

cout << "Creating first level" << endl;

int NumberofChildren = 0;

for (int iCountA = 0; iCountA <= 9; iCountA++)

{

for (int iCountB = iCountA; iCountB <= 9; iCountB++)

{

int iMult = iCountA \* iCountB;

if ((iMult % 10) == (iTarget % 10))

{

LongNumber MultValue = LongNumberMultiplyInt(One, iMult);

if (MultValue <= LNTarget)

{

LongNumber AValue = LongNumberMultiplyInt(One, iCountA);

LongNumber BValue = LongNumberMultiplyInt(One, iCountB);

LongNumber Remainder = LNTarget - MultValue;

// PrintLongNumberLR(Remainder);

bool FactorComplete = false;

if (Remainder == Zero) { FactorComplete = true; }

FactorNode \* FNewNodePt = new FactorNode(FHeadPt, 1, AValue, BValue, MultValue, Remainder, FactorComplete);

// cout << "FNewNodePt : " << &FNewNodePt << endl;

SetParentChild(FHeadPt, \*FNewNodePt);

NumberofChildren++; // increase counter of number of children

if (FactorComplete)

{

PrintFactorNode(FNewNodePt); // if solved in first round then print

NumberofChildren--; // decrease number of children as this is now a leaf

}

}

}

}

}

// Iterate through levels

cout << endl << "Iterating through levels" << endl;

if (NumberofChildren == 0) // no children created in first round so stop

{

cout << "Factor Completed after first round";

return 0;

}

// Create the children

CreateChidNodes(FHeadPt, 1, LNTarget); // Level = 1, Head = 0

cout << "Factoriser END\n";

return 0;

}

void CreateChidNodes(FactorNode \* FNode, int Level, LongNumber LNTarget)

{

int iTarget = LNTarget[Level];

for (int iCountA = 0; iCountA <= 9; iCountA++)

{

for (int iCountB = 0; iCountB <= 9; iCountB++)

{

int iMult = iCountA \* iCountB;

// plus iCountA \* BValue

// pluss iCountB \* AValue

if ((iMult % 10) == (iTarget % 10))

{

LongNumber MultValue = LongNumberMultiplyInt(One, iMult);

if (MultValue <= LNTarget)

{

LongNumber AValue = LongNumberMultiplyInt(One, iCountA);

LongNumber BValue = LongNumberMultiplyInt(One, iCountB);

LongNumber Remainder = LNTarget - MultValue;

// PrintLongNumberLR(Remainder);

bool FactorComplete = false;

if (Remainder == Zero) { FactorComplete = true; }

FactorNode \* FNewNodePt = new FactorNode(FHeadPt, 1, AValue, BValue, MultValue, Remainder, FactorComplete);

// cout << "FNewNodePt : " << &FNewNodePt << endl;

SetParentChild(FHeadPt, \*FNewNodePt);

if (FactorComplete)

{

PrintFactorNode(FNewNodePt); // if solved then print

}

}

}

}

}

}

//int iCurrentNodeChildrenCount = FNode->GetNumberOfChidren();

//while (int iChild = 0 <= iCurrentNodeChildrenCount)

//{

// CurrentNode = GetChildFromParent(CurrentNode, iChild);

// PrintFactorNode(CurrentNode);

// // create nodes for this child

//}

/\*

int iChildrenCount = FHeadPt->GetNumberOfChidren();

for (int iChildrenLoop = 0; iChildrenLoop < iChildrenCount; iChildrenLoop++)

{

cout << "iChildrenLoop : " << iChildrenLoop << endl;

FactorNode\* FactorNodeChild = GetChildFromParent(FHeadPt, iChildrenLoop);

PrintFactorNode(FactorNodeChild);

if (FactorNodeChild->GetFactorComplete() == false) // only do this for incomplete nodes

{

for (int iCountA = 0; iCountA <= 9; iCountA++)

{

for (int iCountB = 0; iCountB <= 9; iCountB++)

{

int iMult = iCountA \* iCountB;

if ((iMult % 10) == (iTarget % 10))

{

LongNumber MultValue = LongNumberMultiplyInt(One, iMult);

if (MultValue <= LNTarget)

{

LongNumber AValue = LongNumberMultiplyInt(One, iCountA);

LongNumber BValue = LongNumberMultiplyInt(One, iCountB);

LongNumber Remainder = LNTarget - MultValue;

// PrintLongNumberLR(Remainder);

bool FactorComplete = false;

if (Remainder == Zero) { FactorComplete = true; }

FactorNode \* FNewNodePt = new FactorNode(FHeadPt, 1, AValue, BValue, MultValue, Remainder, FactorComplete);

// cout << "FNewNodePt : " << &FNewNodePt << endl;

SetParentChild(FHeadPt, \*FNewNodePt);

if (FactorComplete)

{

PrintFactorNode(FNewNodePt); // if solved then print

}

}

}

}

}

}

}

\*/

}