Новосибирск 2023

Выполнил

студент гр. АВТ-310

Шипачева Анна Дмитриевна

Проверил преподаватель

Зеленчук Никита Андреевич

Отчет по лабораторной работе №1

Арифметические задачи

**Министерство образования и науки Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**Новосибирский государственный технический университет**

Кафедра вычислительной техники

Цель работы

1. Научиться разрабатывать алгоритм и реализацию программы с использованием технологии структурного программирования для решения арифметических задач.

Теоретические сведения

Свойства делимости. Такие арифметические процедуры как сокращение дробей, разложение числа на простые множители, определение наименьшего общего кратного, наибольшего общего делителя, поиск простых чисел и т.д. основаны на проверке свойств делимости чисел. Для этой цели используется операция получения остатка от деления «%», число делится на другое число, если остаток от деления равен 0. Нелишне напомнить, что все эти свойства определены для целых чисел, которым в Си соответствуют базовые типы данных int и long.

Работа с цифрами числа. То, что при выводе результата и при написании констант мы наблюдаем число, состоящее их цифр, еще ничего не значит, ибо это есть внешняя форма представления числа. Когда мы используем целую переменную, она представлена в памяти во внутренней (двоичной) форме. То, что с этой формой компьютером выполняются арифметические действия, можно считать «чудом» и не вникать, как он это делает. Отдельные же цифры числа можно получить, используя правила составления числа из цифр в позиционной системе счисления: вес следующей цифры десятичного числа в 10 раз больше текущей. Тогда остаток от деления числа n на 10 можно интерпретировать как значение младшей цифры числа, частное от деления на 10 – как отбрасывание младшей цифры числа, из чего составляется простой цикл получения цифр числа в обратном порядке. Выражение s\*10 «дописывает» к числу 0 справа, а s = s\*10 + k добавляет к нему очередную цифру k.

Поиск полным перебором. Когда речь идет о поиске решения, не стоит забывать, что кроме оригинальных алгоритмов всегда существует «тупой» способ последовательно перебирать все возможные значения, пока не наткнемся на нужное. Возможно, что он будет работать значительно медленнее, но все же приемлемо для наших входных данных. Все арифметические задачи, ребусы, головоломки и т.д. могут решаться путем линейного перебора всех возможных значений чисел с выделением всех, либо первого подходящего варианта. Как правило, такой цикл является внешним в программе поиска. Для поиска первого подходящего проверяемое условие может быть вынесено в заголовок цикла, но только в инверсном виде, поскольку любой цикл имеет в заголовке условие продолжения. Для поиска наименьшего из возможных перебор нужно производить в направлении увеличения проверяемых значений, для поиска наибольшего – в направлении уменьшения. Рассмотрим несколько простых примеров.

Наименьшее общее кратное (НОК) двух целых чисел m и n есть наименьшее натуральное число, которое делится на m и n без остатка, то есть кратно им обоим. Наименьшее общее кратное для нескольких чисел — это наименьшее натуральное число, которое делится на каждое из этих чисел.

Расчетно-графическая часть

Вариант 7

Задание: Делимость чисел. Учитель написал на доске некое натуральное число. После этого первый ученик сказал: «Это число делится на 1». Второй сказал: «Это число делится на 2», ..., 50-й сказал: «Это число делится на 50». И что интересно, только двое из них были неправы. Более того, два неверных утверждения были сделаны подряд одно за другим. Какое наименьшее число мог написать на доске учитель?

1. Основная идея алгоритма

Необходимо с помощью разложения на простые множители найти 2 числа, идущие подряд, таких что при их исключении из чисел 1-50 остальные 48 чисел будут иметь минимальное НОК. Заметим, что утверждения 1-25 заведомо истинные, т.к. иначе возникает противоречие: например, 1 делит любое натуральное число, значит утверждение 1 не может быть ложным; 2 делит любое четное, значит если утверждение 2 ложное, то ложны и 4, 6, и тд., но ложными могут быть только 2 утверждения, причем идущие подряд. Зафиксируем все множители, которые попадут в НОК(1, 2, …, 25)=lcm1, а значит будут являться простыми множителями искомого числа Res. Далее для чисел 26-50 найдем для каждого числа m, на какой множитель нужно домножить lcm1, чтобы получить lcm2=НОК(lcm1, m). Если для некоторого m lcm1=lcm2, то значит простые множители этого числа заведомо попадают в Res, а значит это утверждение истинное, пометим его. Когда закончим нахождение «добавочных» множителей для каждого числа, необходимо найти пары идущих подряд утверждений, не помеченных как истинные. Среди найденных пар нужно найти такие, у которых НОК их «добавочных» множителей максимально, а значит их исключение даст у остальных чисел наименьшее НОК. Окончательно помечаем эти утверждения как ложные, а остальные как истинные. Находим НОК истинных утверждений. Сделаем это не с помощью стандартного алгоритма, а используя знания о тех простых множителях, которые уже имеем. Реализуем программу в общем виде, так чтобы количество проверяемых утверждений не обязательно равнялось 50.

2. Список стандартных фрагментов и необходимых переменных

1) Задать количество утверждений (>2)

#define MAXNUM 50

2) Найти все простые множители до максимального числа, равного количеству утверждений, используя алгоритм «решето Эратосфена»

void findPrimeNumbers();

3) Разложить число на простые множители:

void factorize(int\* muls, int\* degs, int\* size, int num);

int num – число, которое раскладывается на множители;

int\* muls – массив, который хранит простые множители числа num степени >=1;

int\* degs – массив, который хранит соответствующие степени простых множителей числа num;

int\* size – указатель на переменную, хранящую размеры массивов muls и degs.

4) Дополнить таблицу с простыми множителями для НОК новыми множителями, если это необходимо:

void countDegrees(int\* deg, int num);

int\* deg – массив, который хранит данные о степенях множителей НОК нескольких чисел;

int num – число, которое хотим добавить к этим числам и найти для них НОК. Если все простые множители num в нужной степени уже отмечены в deg, то deg не изменяются, иначе там изменяется степень нужного множителя.

5) Посчитать, на какой множитель нужно домножить уже найденное НОК, чтобы новое число его делило:

int checkMuls(int\* deg, int num);

int\* deg – массив, который хранит данные о степенях множителей НОК нескольких чисел;

int num – число, которое хотим добавить к этим числам и найти для них НОК. Если все простые множители num в нужной степени уже отмечены в deg, то функция возвращает 1, иначе возвращает множитель, на который нужно домножить старое НОК, чтобы получить новое.

6) Посчитать НОК двух чисел:

int findLcm(int a, int b);

7) Переменные:

int primeNumbers[MAXNUM] – массив, хранящий простые числа до MAXNUM (=50);

int sizeArrPrime – размер массива primeNumbers;

int i – счетчик;

int numbers[2][MAXNUM] – массив, который хранит флаг, что данное число (индекс+1) делит искомое, а также «добавочный» множитель;

int degrees[MAXNUM] – массив, который хранит степени множителей для НОК. Множитель=индекс+1;

int check – результат функции checkMuls;

int flag – используется как флаг для поиска пары идущих подряд не истинных утверждений;

int pair[2] – массив, в котором первый элемент хранит индекс первого элемента пары, а второй элемент хранит НОК элементов пары;

double res – искомое число. Используем double, т.к. в int возникает переполнение.

3. Текст программы с комментариями

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#define MAXNUM 50 //любое число >2

void findPrimeNumbers(); //решето Эратосфена, найти простые числа до //MAXNUM

void countDegrees(int\* deg, int num); //посчитать степени множителей, //которые входят в НОК

int checkMuls(int\* deg, int num); //проверить, делит ли новое число уже //имеющийся НОК

int findLcm(int a, int b);//найти НОК двух чисел

int primeNumbers[MAXNUM];

int sizeArrPrime = 0;

int main() {

findPrimeNumbers(); //найти простые числа до MAXNUM

int i;

int numbers[2][MAXNUM] = { 0 }; //хранятся флаги, что данное число //делит искомое и "добавочные" множители

for (i = 0; i < MAXNUM / 2; i++) { //проставить флаг, что первая //половина чисел делит искомое

numbers[0][i] = 1;

numbers[1][i] = 1;

}

int degrees[MAXNUM] = { 0 }; //хранятся степени множителей для НОК

degrees[1 - 1] = 1;

for (i = 2; i <= MAXNUM / 2; i++) {

//посчитать какие множители и в какой степени есть в первой //половине чисел

countDegrees(degrees, i);

}

for (i = MAXNUM / 2; i < MAXNUM; i++) { //какой множитель нужно //добавить к уже имеющемуся "набору"

int check = checkMuls(degrees, i + 1); //множителей, чтобы на //это число делилось искомое

if (check == 1) {//это число уже делит искомое

numbers[0][i] = numbers[1][i] = 1;

}

else { //чтобы это число делило искомое, нужно добавить //множитель check

numbers[0][i] = 0;

numbers[1][i] = check;

}

}

int flag = 0;

int pair[2] = { 0 }; //первый элемент-стартовый элемент пары; второй //элемент - НОК "добавочных" множителей пары

for (i = MAXNUM / 2; i < MAXNUM; i++) {

if (numbers[0][i] == 0) { //ищем два числа подряд, не делители //искомого

if (flag == 0)

flag = 1;

else { //нашли пару

int Lcm = findLcm(numbers[1][i], numbers[1][i - 1]);

if (pair[0] == 0 || Lcm > pair[1]) { //сравниваем //новую пару с предыдущей если она была

pair[0] = i-1;

pair[1] = Lcm;

}

}

}

else

flag = 0;

}

if (pair[0] == 0) {

printf("No pair found");

return 0;

}

//пометить, что эту пару не используем

numbers[0][pair[0]] = -1;

numbers[1][pair[0]] = 1;

numbers[0][pair[0] + 1] = -1;

numbers[1][pair[0] + 1] = 1;

//дополнить информацию по степеням множителей в НОК (искомом числе)

for (i = MAXNUM / 2; i < MAXNUM; i++) {

if (numbers[0][i] >= 0) {

countDegrees(degrees, i + 1);

}

}

//подсчитать НОК

double res = 1.0;

printf("%d statements\n", MAXNUM);

printf("Result: 1 ");

for (i = 1; i < MAXNUM; i++) {

for (int j = 0; j < degrees[i]; j++) {

res \*= (i + 1) \* 1.0;

}

printf("\* %d^%d ", i + 1, degrees[i]);

}

printf("\n= %.0lf", res);

}

void findPrimeNumbers() { //найти простые числа меньше или равные MAXNUM, //решето Эратосфена

int arr[MAXNUM] = { 0 };

int i;

for (i = 1; i < MAXNUM; i++) {

arr[i] = 1;

}

for (i = 1; i < MAXNUM / 2; i++) {

if (arr[i] == 1) {

for (int j = i + (i + 1); j < MAXNUM; j += i + 1) {

arr[j] = 0;

}

}

}

for (i = 1; i < MAXNUM; i++) {

if (arr[i] == 1) {

primeNumbers[sizeArrPrime] = i + 1;

sizeArrPrime++;

}

}

}

void factorize(int\* muls, int\* degs, int\* size, int num) { //разложение //числа на простые множители

int i = 0, j = 1;

int count = 0;

int mul = 0;

\*size = 0;

while (i < sizeArrPrime) {

if (num % primeNumbers[i] == 0) {

mul = primeNumbers[i];

count++;

num = num / mul;

}

else {

if (mul != 0) {

muls[\*size] = mul;

degs[\*size] = count;

\*size = \*size + 1;

mul = 0;

count = 0;

}

i++;

}

}

}

void countDegrees(int\* deg, int num) { //дополнить таблицу по уже //имеющимся множителям для НОК

int muls[MAXNUM] = { 0 };

int degs[MAXNUM] = { 0 };

int size = 0;

factorize(muls, degs, &size, num);

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

if (degs[i] > deg[muls[i] - 1]) {

deg[muls[i] - 1] = degs[i];

}

}

return;

}

int checkMuls(int\* deg, int num) { //проверить, что все множители данного //числа есть в НОК (возвращает 1) иначе возвращает множитель, которого //не хватает

int muls[MAXNUM] = { 0 };

int degs[MAXNUM] = { 0 };

int size = 0;

int res = 1;

factorize(muls, degs, &size, num);

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

if (deg[muls[i] - 1] < degs[i]) {//если такого множителя не было //в первой половине чисел

for (int j = 0; j < degs[i] - deg[muls[i] - 1]; j++) {

res \*= muls[i];

}

}

}

return res;

}

int findLcm(int a, int b) { //найти НОК

int res = a \* b;

while (a % b != 0 && b % a != 0) {

if (a > b) {

a = a % b;

}

else {

b = b % a;

}

}

if (a < b)

res = res / a;

else

res = res / b;

return res;

}

4. Результат работы

50 statements

Result: 1 \* 2^4 \* 3^3 \* 4^0 \* 5^2 \* 6^0 \* 7^2 \* 8^0 \* 9^0 \* 10^0 \* 11^1 \* 12^0 \* 13^1 \* 14^0 \* 15^0 \* 16^0 \* 17^1 \* 18^0 \* 19^1 \* 20^0 \* 21^0 \* 22^0 \* 23^1 \* 24^0 \* 25^0 \* 26^0 \* 27^0 \* 28^0 \* 29^1 \* 30^0 \* 31^0 \* 32^0 \* 33^0 \* 34^0 \* 35^0 \* 36^0 \* 37^1 \* 38^0 \* 39^0 \* 40^0 \* 41^1 \* 42^0 \* 43^1 \* 44^0 \* 45^0 \* 46^0 \* 47^1 \* 48^0 \* 49^0 \* 50^0

= 49984588778161233920

Вариант 15

Задание: Числовой ребус. В этом примере на умножение присутствуют все цифры от 0 до 9, причем каждая цифра встречается только однажды (цифры в промежуточных выкладках в расчет не идут). Решите этот пример. Чтобы вам было от чего отправляться, мы вписали во второй сомножитель одну цифру. XXX \* X5 = XXXXX

1. Основная идея алгоритма

Линейным перебором составлять комбинации из десяти цифр, соответствующие условиям:

-первая, вторая и третья позиции соответствуют первому числу (трехзначному)

-четвертая и пятая позиции соответствуют второму числу(двузначному)

-шестая, седьмая, восьмая, девятая, десятая позиции соответствуют третьему числу (пятизначному)

-на пятой позиции всегда стоит цифра 5. Значит, на десятой позиции стоит либо 5, либо 0, но т.к. 5 уже занято, значит на десятой позиции всегда стоит 0. Из этого следует, что на третьей позиции стоит либо 2, либо 4, либо 6, либо 8.

Цифра для позиции берется из «пула» цифр, в котором помечается, если какая-то цифра занята. Это исключает комбинации с повторяющимися цифрами.

После составления комбинации проверить ее на условие:

-произведение первого числа на второе должно равняться третьему числу

2. Список стандартных фрагментов и необходимых переменных

1) Цикл для заполнения очередной позиции в комбинации. Восемь вложенных циклов аналогичного вида.

for (a = 0; a < 10; a++)

2) Проверка условия первое\*второе=третье.

int checkNums(int\* firstArr, int\* secondArr, int\* thirdArr)

3) Перевод в число из набора цифр

int fromDigits(int\* num, int count)

Переменные:

int firstNum[3] – массив, который хранит цифры первого числа;

int secondNum[2] - массив, который хранит цифры второго числа;

int thirdNum[5] - массив, который хранит цифры третьего числа;

int digits[10] – пул цифр;

int a, b, c, d, e, f, g, h, i – счетчики для вложенных циклов, заполняющих позиции числа цифрами.

3. Текст программы с комментариями

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

int checkNums(int \* firstArr, int \* secondArr, int \* thirdArr);

//проверить что первое\*второе=третье

int main() {

int firstNum[3]={-1,-1,-1}; //первое число

int secondNum[2]={-1,-1}; //второе число

int thirdNum[5]={-1,-1,-1,-1,-1}; //третье число

int digits[10] = { 0 };

int a, b, c, d, e, f, g, h, i;

secondNum[1] = 5; //вторая цифра второго числа

thirdNum[4] = 0;

digits[0] = 1;

digits[5] = 1;

//создание новых последовательностей

for (a = 0; a < 10; a++) { //заполнение первой цифры первого числа

if (digits[a] != 1) {

firstNum[0] = a;

digits [a] = 1;

for (b = 0; b < 10; b++) { //заполнение второй цифры первого //числа

if (digits[b] != 1) {

firstNum[1] = b;

digits[b] = 1;

for (c = 2; c < 10; c += 2) { //заполнение третьей цифры //первого числа

if (digits[c] != 1) {

firstNum[2] = c;

digits[c] = 1;

for (d = 0; d < 10; d++) { //заполнение первой цифры //второго числа

if (digits[d] != 1) {

secondNum[0] = d;

digits[d] = 1;

for (e = 0; e < 10; e++) {//заполнение первой //цифры третьего числа

if (digits[e] != 1) {

thirdNum[0] = e;

digits[e] = 1;

for (f = 0; f < 10; f++) {//заполнение //второй цифры третьего числа

if (digits[f] != 1) {

thirdNum[1] = f;

digits[f] = 1;

for (g = 0; g < 10; g++) { //заполнение //третьей цифры третьего цисла

if (digits[g] != 1) {

thirdNum[2] = g;

digits[g] = 1;

for (h = 0; h < 10; h++) {

if (digits[h] != 1) {

thirdNum[3] = h;

digits[h] = 1;

//проверка чисел

if (checkNums(firstNum, secondNum, thirdNum)) {

printf("Solution: %d%d%d \* %d%d = %d%d%d%d%d", firstNum[0], firstNum[1], firstNum[2],

secondNum[0], secondNum[1],

thirdNum[0], thirdNum[1], thirdNum[2], thirdNum[3], thirdNum[4]);

return 0;

}

digits[h] = 0;

}

}

digits[g] = 0;

}

}

digits[f] = 0;

}

}

digits[e] = 0;

}

}

digits[d] = 0;

}

}

digits[c] = 0;

}

}

digits[b] = 0;

}

}

digits[a] = 0;

}

}

printf("No solution found");

return 0;

}

int fromDigits(int\* num, int count) {//перевод из массива в число

int res = 0;

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

res = res \* 10;

res = res + num[i];

}

return res;

}

int checkNums(int\* firstArr, int\* secondArr, int\* thirdArr) {

int first = fromDigits(firstArr, 3);

int second = fromDigits(secondArr, 2);

int third = fromDigits(thirdArr, 5);

int res = first \* second;

return res == third;

}

4. Результат работы

Solution: 396 \* 45 = 17820

Вывод

В ходе выполнения лабораторной работы были разработаны алгоритмы с использованием технологии структурного программирования. В программе были использованы циклы for, while, функции нахождения простых чисел, разложения на множители, для работы с цифрами числа.