# Задание

Написать программу, на языке С++ в среде Visual Studio для определения, является ли множество С объединением множеств A и B (AB), пересечением множеств (AB), разностью множеств А и B (A\B), разностью множеств В и А (В\А)

Однородные вычислительные структуры или среды (ОВС), как правило, относятся к классу ОКМД (согласно классификации Флинна) и представляют собой регулярную структуру из однотипных процессорных элементов (ПЭ).

Каждый ПЭ, в зависимости от типа ОВС, может как обладать алгоритмически полным набором операций, так и реализовывать один вид операций, жестко заданный в структуре микросхемы на этапе проектирования, а также выполнять операции обмена или взаимодействия с другими ПЭ.

# OpenMP

Среди специалистов, занимающихся параллельными вычислениями, популярна шутка «Параллельные вычисления — технология будущего... и так будет всегда». Эта шутка не теряет актуальность уже несколько десятилетий. Аналогичные настроения были распространены в сообществе разработчиков архитектур компьютеров, обеспокоенном тем, что скоро будет достигнут предел тактовой частоты процессоров, однако частоты процессоров продолжают повышаться, хотя гораздо медленнее, чем раньше. Сплав оптимизма специалистов по параллельным вычислениям и пессимизма архитекторов систем способствовал появлению революционных многоядерных процессоров.

Главные производители процессоров сместили акцент с повышения тактовых частот на реализацию параллелизма в самих процессорах за счет использования многоядерной архитектуры. Идея проста: интегрировать в один процессор более одного ядра. Система, включающая процессор с двумя ядрами, по сути, не отличается от двухпроцессорного компьютера, а система с четырехядерным процессором — от четырехпроцессорного. Этот подход позволяет избежать многих технологических проблем, связанных с повышением тактовых частот, и создавать при этом более производительные процессоры.

Все это прекрасно, но если ваше приложение не будет использовать несколько ядер, его быстродействие никак не изменится. Именно здесь и вступает в игру технология OpenMP, которая помогает программистам на C++ быстрее создавать многопоточные приложения.

Стандарт OpenMP был разработан в 1997 г. как API, ориентированный на написание портируемых многопоточных приложений. Сначала он был основан на языке Fortran, но позднее включил в себя и C/C++. Последняя версия OpenMP — 2.0;  ее полностью поддерживает Visual C++ 2005. Стандарт OpenMP поддерживается и платформой Xbox 360.

в Visual C++ 2005 параметр компилятора /openmp. (Вы можете активизировать директивы OpenMP на страницах свойств проекта, выбрав Configuration Properties, C/C++, Language и изменив значение свойства OpenMP Support.) Встретив параметр /openmp, компилятор определяет символ \_OPENMP, с помощью которого можно выяснить, включены ли средства OpenMP. Для этого достаточно написать #ifndef \_OPENMP.

OpenMP связывается с приложениями через библиотеку импорта vcomp.lib. Соответствующая библиотека периода выполнения называется vcomp.dll. Отладочные версии библиотек импорта и периода выполнения (vcompd.lib и vcompd.dll соответственно) поддерживают дополнительные сообщения об ошибках, генерируемых при некоторых недопустимых операциях. Имейте в виду, что Visual C++ не поддерживает статическое связывание с библиотекой OpenMP периода выполнения, хотя в версии для Xbox 360 это поддерживается.

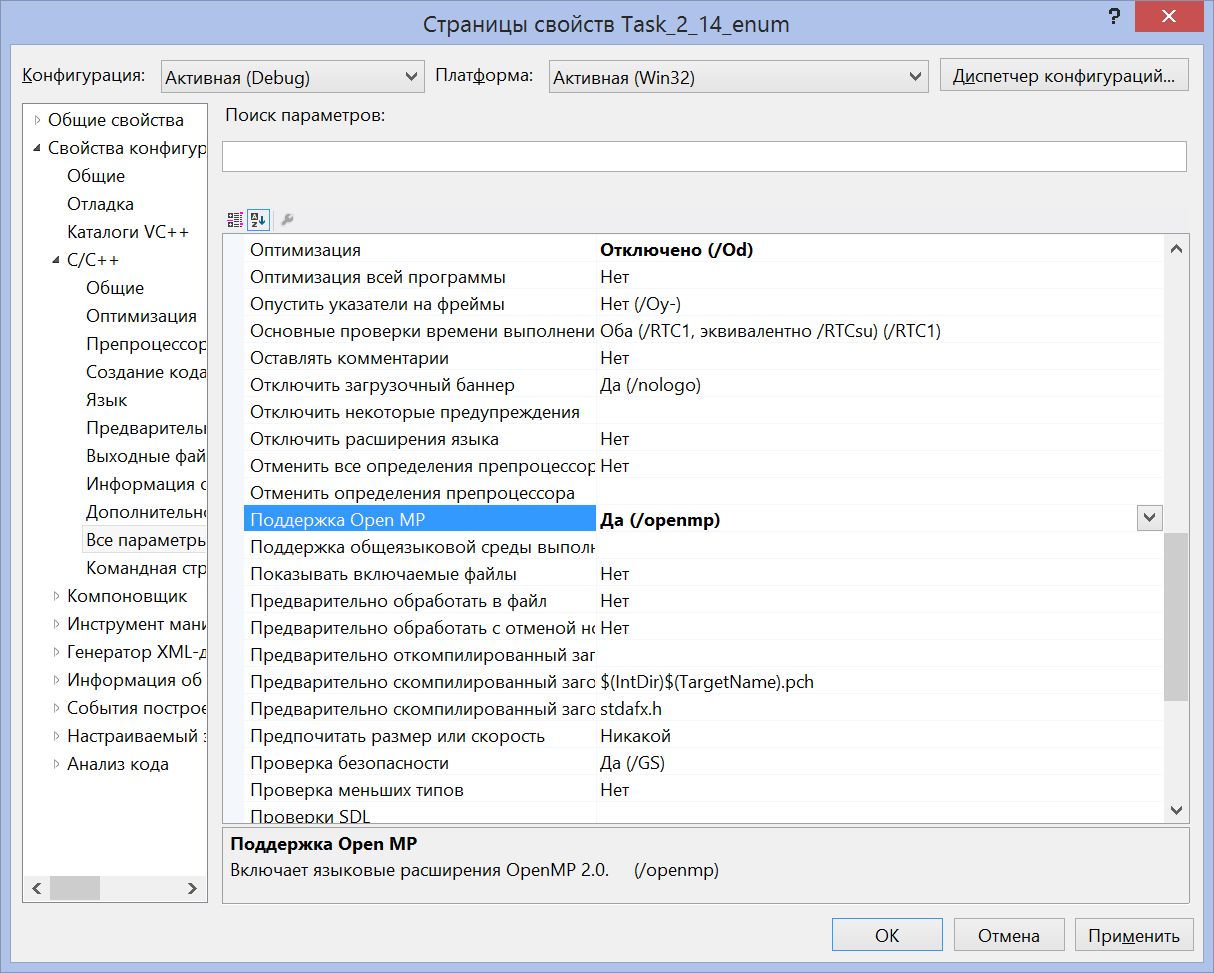
Работа OpenMP-приложения начинается с единственного потока — основного. В приложении могут содержаться параллельные регионы, входя в которые, основной поток создает группы потоков (включающие основной поток). В конце параллельного региона группы потоков останавливаются, а выполнение основного потока продолжается. В параллельный регион могут быть вложены другие параллельные регионы, в которых каждый поток первоначального региона становится основным для своей группы потоков. Вложенные регионы могут в свою очередь включать регионы более глубокого уровня вложенности.

OpenMP прост в использовании и включает лишь два базовых типа конструкций: директивы pragma и функции исполняющей среды OpenMP. Директивы pragma, как правило, указывают компилятору реализовать параллельное выполнение блоков кода. Все эти директивы начинаются с #pragma omp. Как и любые другие директивы pragma, они игнорируются компилятором, не поддерживающим конкретную технологию — в данном случае OpenMP.

Функции OpenMP служат в основном для изменения и получения параметров среды. Кроме того, OpenMP включает API-функции для поддержки некоторых типов синхронизации. Чтобы задействовать эти функции библиотеки OpenMP периода выполнения (исполняющей среды), в программу нужно включить заголовочный файл omp.h. Если вы используете в приложении только OpenMP-директивы pragma, включать этот файл не требуется.

Для реализации параллельного выполнения блоков приложения нужно просто добавить в код директивы pragma и, если нужно, воспользоваться функциями библиотеки OpenMP периода выполнения.

OpenMP поддерживает директивы parallel, for, parallel for, section, sections, single, master, critical, flush, ordered и atomic, которые определяют или механизмы разделения работы или конструкции синхронизации.



# Операции со множествами

Элементы множества задаются целочисленными идентификаторами

Составляем полный список всех используемых идентификаторов элементов множеств

Сопоставляем каждому множеству двоичный вектор

Двоичный вектор флагов объединения множеств A и B x1=a or b

Двоичный вектор флагов пересечения множеств A и B x2=a and b

Двоичный вектор флагов разности множеств А и B x3=a sub b

Двоичный вектор флагов разности множеств B и A x4=b sub a

|  |  |  |
| --- | --- | --- |
| **or** | **0** | **1** |
| **0** | 0 | 1 |
| **1** | 1 | 1 |

|  |  |  |
| --- | --- | --- |
| **and** | **0** | **1** |
| **0** | 0 | 0 |
| **1** | 0 | 1 |

|  |  |  |
| --- | --- | --- |
| **sub** | **0** | **1** |
| 0 | 0 | 0 |
| 1 | 1 | 0 |

|  |  |  |
| --- | --- | --- |
| **xor** | **0** | **1** |
| **0** | 0 | 1 |
| **1** | 1 | 0 |

Двоичный вектор x==y тогда и только тогда, когда count(x xor y)==0 где count – количество ненулевых элементов в векторе

Программа работает в 2-х режимах:

* с выводом результатов работы на экран (для демонстрации работоспособности);
* без вывода результатов работы на экран, но с определением времени, затрачиваемого на вычисления;

Листинг кода

#include "Header.hpp"

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Обнуление двоичного вектора

Однородные вычислительные структуры или среды (ОВС), как правило,

относятся к классу ОКМД (согласно классификации Флинна)

и представляют собой регулярную структуру из однотипных процессорных элементов (ПЭ).

Каждый ПЭ, в зависимости от типа ОВС, может как обладать алгоритмически полным набором

операций, так и реализовывать один вид операций, жестко заданный в структуре микросхемы

на этапе проектирования, а также выполнять операции обмена или взаимодействия с другими ПЭ.

OpenMP поддерживает директивы parallel, for, parallel for, section, sections, single, master, critical, flush, ordered и atomic,

которые определяют или механизмы разделения работы или конструкции синхронизации.

Общие переменные доступны всем потокам из группы, поэтому изменения таких переменных в одном потоке видимы другим потокам

в параллельном регионе. Что касается частных переменных, то каждый поток из группы располагает их отдельными экземплярами,

поэтому изменения таких переменных в одном потоке никак не сказываются на их экземплярах,

принадлежащих другим потокам.

частными являются индексы параллельных циклов for.

vector - указатель на массив

count - количество элементов

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void clear(int \*vector, int count)

{

#pragma omp parallel for

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

vector[i]=0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Число ненулевых элементов двоичного вектора

Суммирование элементов требует дополнительных обработок для параллельных вычислений

Поэтому используется непараллельная версия

vector - указатель на массив

count - количество элементов

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int sum(int \*vector, int count)

{

int s=0;

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

s+=vector[i];

return s;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Булева операция AND элементов двоичных векторов

Однородные вычислительные структуры или среды (ОВС), как правило,

относятся к классу ОКМД (согласно классификации Флинна)

и представляют собой регулярную структуру из однотипных процессорных элементов (ПЭ).

Каждый ПЭ, в зависимости от типа ОВС, может как обладать алгоритмически полным набором

операций, так и реализовывать один вид операций, жестко заданный в структуре микросхемы

на этапе проектирования, а также выполнять операции обмена или взаимодействия с другими ПЭ.

OpenMP поддерживает директивы parallel, for, parallel for, section, sections, single, master, critical, flush, ordered и atomic,

которые определяют или механизмы разделения работы или конструкции синхронизации.

Общие переменные доступны всем потокам из группы, поэтому изменения таких переменных в одном потоке видимы другим потокам

в параллельном регионе. Что касается частных переменных, то каждый поток из группы располагает их отдельными экземплярами,

поэтому изменения таких переменных в одном потоке никак не сказываются на их экземплярах,

принадлежащих другим потокам.

частными являются индексы параллельных циклов for.

vector1 - указатель на массив

vector2 - указатель на массив

result - указатель на массив

count - количество элементов

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void and(int \*vector1, int \*vector2,int \*result, int count)

{

#pragma omp parallel for

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

result[i]=vector1[i]&vector2[i];

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Булева операция OR элементов двоичных векторов

Однородные вычислительные структуры или среды (ОВС), как правило,

относятся к классу ОКМД (согласно классификации Флинна)

и представляют собой регулярную структуру из однотипных процессорных элементов (ПЭ).

Каждый ПЭ, в зависимости от типа ОВС, может как обладать алгоритмически полным набором

операций, так и реализовывать один вид операций, жестко заданный в структуре микросхемы

на этапе проектирования, а также выполнять операции обмена или взаимодействия с другими ПЭ.

OpenMP поддерживает директивы parallel, for, parallel for, section, sections, single, master, critical, flush, ordered и atomic,

которые определяют или механизмы разделения работы или конструкции синхронизации.

Общие переменные доступны всем потокам из группы, поэтому изменения таких переменных в одном потоке видимы другим потокам

в параллельном регионе. Что касается частных переменных, то каждый поток из группы располагает их отдельными экземплярами,

поэтому изменения таких переменных в одном потоке никак не сказываются на их экземплярах,

принадлежащих другим потокам.

частными являются индексы параллельных циклов for.

vector1 - указатель на массив

vector2 - указатель на массив

result - указатель на массив

count - количество элементов

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void or(int \*vector1, int \*vector2,int \*result, int count)

{

#pragma omp parallel for

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

result[i]=vector1[i]|vector2[i];

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Булева операция XOR элементов двоичных векторов

Однородные вычислительные структуры или среды (ОВС), как правило,

относятся к классу ОКМД (согласно классификации Флинна)

и представляют собой регулярную структуру из однотипных процессорных элементов (ПЭ).

Каждый ПЭ, в зависимости от типа ОВС, может как обладать алгоритмически полным набором

операций, так и реализовывать один вид операций, жестко заданный в структуре микросхемы

на этапе проектирования, а также выполнять операции обмена или взаимодействия с другими ПЭ.

OpenMP поддерживает директивы parallel, for, parallel for, section, sections, single, master, critical, flush, ordered и atomic,

которые определяют или механизмы разделения работы или конструкции синхронизации.

Общие переменные доступны всем потокам из группы, поэтому изменения таких переменных в одном потоке видимы другим потокам

в параллельном регионе. Что касается частных переменных, то каждый поток из группы располагает их отдельными экземплярами,

поэтому изменения таких переменных в одном потоке никак не сказываются на их экземплярах,

принадлежащих другим потокам.

частными являются индексы параллельных циклов for.

vector1 - указатель на массив

vector2 - указатель на массив

result - указатель на массив

count - количество элементов

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void xor(int \*vector1, int \*vector2,int \*result, int count)

{

#pragma omp parallel for

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

result[i]=vector1[i]^vector2[i];

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Булева операция SUB элементов двоичных векторов

Однородные вычислительные структуры или среды (ОВС), как правило,

относятся к классу ОКМД (согласно классификации Флинна)

и представляют собой регулярную структуру из однотипных процессорных элементов (ПЭ).

Каждый ПЭ, в зависимости от типа ОВС, может как обладать алгоритмически полным набором

операций, так и реализовывать один вид операций, жестко заданный в структуре микросхемы

на этапе проектирования, а также выполнять операции обмена или взаимодействия с другими ПЭ.

OpenMP поддерживает директивы parallel, for, parallel for, section, sections, single, master, critical, flush, ordered и atomic,

которые определяют или механизмы разделения работы или конструкции синхронизации.

Общие переменные доступны всем потокам из группы, поэтому изменения таких переменных в одном потоке видимы другим потокам

в параллельном регионе. Что касается частных переменных, то каждый поток из группы располагает их отдельными экземплярами,

поэтому изменения таких переменных в одном потоке никак не сказываются на их экземплярах,

принадлежащих другим потокам.

частными являются индексы параллельных циклов for.

vector1 - указатель на массив

vector2 - указатель на массив

result - указатель на массив

count - количество элементов

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void sub(int \*vector1, int \*vector2,int \*result, int count)

{

#pragma omp parallel for

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

result[i]=vector1[i]&(vector1[i]^vector2[i]);

}

#include "Header.hpp"

using namespace std;

// 14. Определить, является ли множество С

// объединением множеств A и B (A B),

// пересечением множеств A и B (A B),

// разностью множеств А и B (A\B),

// разностью множеств В и А (В\А).

// элементы множества задаются целочисленными идентификаторами

int main()

{

int \*a; // идентификаторы элементов множества A

int \*b; // идентификаторы элементов множества B

int \*c; // идентификаторы элементов множества C

int na; // количество элементов множества A

int nb; // количество элементов множества B

int nc; // количество элементов множества C

int demo; // режим программы

setlocale(LC\_ALL, "Russian");

cout << "\tОпределить, является ли множество С" << endl;

cout << "\tобъединением множеств A и B" << endl;

cout << "\tпересечением множеств A и B" << endl;

cout << "\tразностью множеств А и B" << endl;

cout << "\tразностью множеств В и А" << endl;

cout << "\tВведите количество элементов множества A:" << endl; cin >> na;

a = new int[na+1]; // +1 для случая пустого множества

cout << "\tВведите идентификаторы элементов множества A["<<na<<"]:" << endl;

for(int i=0;i<na;i++) cin>>a[i];

cout << "\tВведите количество элементов множества B:" << endl; cin >> nb;

b = new int[nb+1]; // +1 для случая пустого множества

cout << "\tВведите идентификаторы элементов множества B["<<nb<<"]:" << endl;

for(int i=0;i<nb;i++) cin>>b[i];

cout << "\tВведите количество элементов множества C:" << endl; cin >> nc;

c = new int[nc+1]; // +1 для случая пустого множества

cout << "\tВведите идентификаторы элементов множества C["<<nc<<"]:" << endl;

for(int i=0;i<nc;i++) cin>>c[i];

cout << "\tДемонстрационный режим програмы (0 - замер времени, 1 - демонстрация работоспособности):" << endl; cin >> demo;

clock\_t t = clock();

// Составляем полный список всех используемых идентификаторов элементов множеств

int \*ids = new int[na+nb+nc+1]; // Список всех используемых идентификаторов +1 для случая пустого множества

int count=0; // Количество используемых идентификаторов

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

int id = a[i];

int index=count;

while(index-->0) if(ids[index]==id) break;

if(index==-1) ids[count++]=id;

}

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

int id = b[i];

int index=count;

while(index-->0) if(ids[index]==id) break;

if(index==-1) ids[count++]=id;

}

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

int id = c[i];

int index=count;

while(index-->0) if(ids[index]==id) break;

if(index==-1) ids[count++]=id;

}

// Сопоставляем каждому множеству двоичный вектор

int \*va = new int[count+1]; // Двоичный вектор флагов множества A

int \*vb = new int[count+1]; // Двоичный вектор флагов множества B

int \*vc = new int[count+1]; // Двоичный вектор флагов множества C

clear(va, count); // используется параллельная обработка элементов массива

clear(vb, count); // используется параллельная обработка элементов массива

clear(vc, count); // используется параллельная обработка элементов массива

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

int id = a[i];

int index=count;

while(index-->0) if(ids[index]==id) break;

va[index]=1;

}

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

int id = b[i];

int index=count;

while(index-->0) if(ids[index]==id) break;

vb[index]=1;

}

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

int id = c[i];

int index=count;

while(index-->0) if(ids[index]==id) break;

vc[index]=1;

}

if(demo!=0){

cout << "\tВектор a:"; for(int i=0;i<count;i++) cout<<va[i]; cout << endl;

cout << "\tВектор b:"; for(int i=0;i<count;i++) cout<<vb[i]; cout << endl;

cout << "\tВектор c:"; for(int i=0;i<count;i++) cout<<vc[i]; cout << endl;

}

int \*x1 = new int[count+1]; // Двоичный вектор флагов объединения множеств A и B

int \*x2 = new int[count+1]; // Двоичный вектор флагов пересечения множеств A и B

int \*x3 = new int[count+1]; // Двоичный вектор флагов разности множеств А и B

int \*x4 = new int[count+1]; // Двоичный вектор флагов разности множеств B и A

or(va, vb, x1, count); // используется параллельная обработка элементов массива

and(va, vb, x2, count); // используется параллельная обработка элементов массива

sub(va, vb, x3, count); // используется параллельная обработка элементов массива

sub(vb, va, x4, count); // используется параллельная обработка элементов массива

if(demo!=0){

cout << "\tВектор a|b:"; for(int i=0;i<count;i++) cout<<x1[i]; cout << endl;

cout << "\tВектор a&b:"; for(int i=0;i<count;i++) cout<<x2[i]; cout << endl;

cout << "\tВектор a-b:"; for(int i=0;i<count;i++) cout<<x3[i]; cout << endl;

cout << "\tВектор b-a:"; for(int i=0;i<count;i++) cout<<x4[i]; cout << endl;

}

int \*y = new int[count+1];

xor(vc,x1,y, count);

if(sum(y,count)==0) cout << "a|b == c"<< endl;

else cout << "a|b != c"<< endl;

xor(vc,x2,y, count);

if(sum(y,count)==0) cout << "a&b == c"<< endl;

else cout << "a&b != c"<< endl;

xor(vc,x3,y, count);

if(sum(y,count)==0) cout << "a-b == c"<< endl;

else cout << "a-b != c"<< endl;

xor(vc,x4,y, count);

if(sum(y,count)==0) cout << "b-a == c"<< endl;

else cout << "b-a != c"<< endl;

delete y;

delete x1;

delete x2;

delete x3;

delete x4;

delete va;

delete vb;

delete vc;

delete ids;

t = clock()-t;

double seconds = ((double)t)/CLOCKS\_PER\_SEC;

cout << "\tВремя выполнения: " << seconds << "sec" << endl;

delete a;

delete b;

delete c;

getchar();

getchar();

return 0;

}

# Контрольные примеры работы программы

Пример 1.

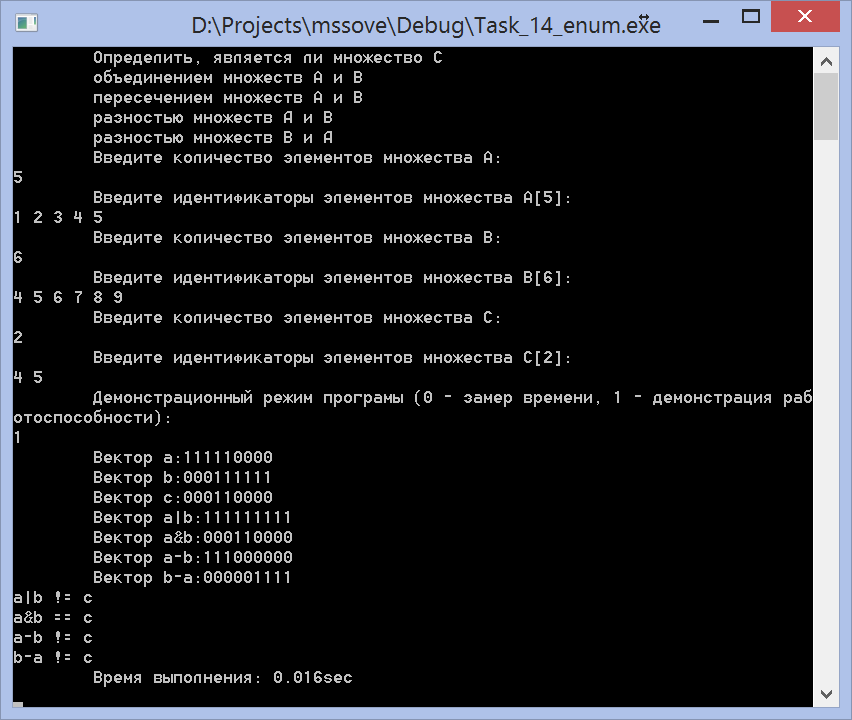


Рис.1..

# Заключение

В ходе данной лабораторной был изучены операции над множествами и булевыми векторами. Была сделана программная реализация данного алгоритма. Для данной программной реализации были проведены ряд тестов, показывающие правильность работы алгоритма.