**СЕМИНАР 6. Перегрузка операций. Alpha. Часть 2**

**Разбор программы**

Разработать ООП для подсчета числа различных и совпадающих латинских букв без различия регистра в любой паре их слов, которые заданы аргументами командной строки ее вызова. Программная реализация вычислений должна быть основана на разработке класса подмножества латинских букв, состав которого кодируется в приватном поле двоичными разрядами целого числа без знака. Конструктор класса должен обеспечивать его инициализацию по заданной или пустой строке символов. Кроме того, должна быть предусмотрена компонентная перегрузка операторов “( )” и “,” для эффективного вычисления расстояния Хемминга и скалярного произведения бинарных кодов заданных строк при количественной оценке их различия и совпадения, соответственно. При этом операнды скалярного произведения следует заключать в круглые скобки, а скобочная перегрузка Хемминга должна вызываться пустым набором H (из косметических соображений и по формальным правилам). Также в обоих операциях должен применяться эффективный метод подсчета единичных разрядов. Результаты вычислений должны отображаться строками стандартного вывода. Отображение вычислительных формул должны обеспечивать дружественная перегрузка оператора << класса потока стандартного вывода и(или) компонентный оператор преобразования типа латинского набора в адрес строки его символов.

**Пример результатов работы программы:**

$ AOO XeniX Uetrix

H(einx, eirtux) = 4

(EINX, EIRTUX) = 3

**Комментарии к постановке и решению задачи:**

**Расстояние Хемминга** – число позиций, в которых соответствующие символы двух [слов](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B2%D0%BE_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) ***одинаковой*** длины различны.

Пример расчета расстояния Хемминга:



Подсчитываемые символы выделены в словах цветом.

В программе происходит сравнение двух векторов длины 32 разряда (unsigned int = 4 байта = 32 бита), таким образом обеспечивается условие равенства длин сравниваемых векторов. Кодировка исходных слов обеспечивает, что одноименные буквы будут находиться в идентичных позициях исходных векторов.

Так кодировка слов XeniX и Uetrix будет выглядеть следующим образом:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | z | y | x | w | u | v | t | s | r | q | p | o | n | m | l | k | j | i | h | g | f | e | d | c | b | a |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

Красным выделены позиции, по которым слова не совпадают и учитываются при подсчете, а желтым – по которым слова совпадают между собой.

Для определения таковых позиций в программе используется оператор ^ - побитовое исключающее или, которое имеет следующую таблицу истинности (1 в тех позициях, в которых значения бит различны):



**Скалярное произведение** в контексте данного задания определяет количество позиций по которым векторы совпадают между собой.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | z | y | x | w | u | v | t | s | r | q | p | o | n | m | l | k | j | i | h | g | f | e | d | c | b | a |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

Красным выделены позиции, по которым слова совпадают и учитываются при подсчете, а желтым – по которым слова не совпадают между собой.

Для определения таковых позиций в программе используется оператор & - побитовое И, которое имеет следующую таблицу истинности (1 в тех позициях, в которых оба бита имеют значение 1):



В методе **быстрого подсчета единичных разрядов** реализована следующая идея.

Если число уменьшить на единицу, то в его бинарном коде самый правый единичный разряд обратится в 0, все стоящие правее него разряды станут единичными, а разряды, стоящие слева, останутся неизменными. Это связано с необходимостью увеличения числа разрядов при увеличении максимального для заданного количества разрядов числа

( 1 1 1 1 2 + 1 --–> 1 0 0 0 0 2 )

( 1 0 0 0 0 2 – 1 --–> 1 1 1 1 2 )

Например,

140010 = 1 0 1 0 1 1 1 1 0 0 0 2 - 1 =

139910 = 1 0 1 0 1 1 1 0 1 1 1 2

Если впоследствии выполнить операцию побитового И между числом и числом, на единицу меньше его, то все разряды, начиная с последнего единичного в исходном числе и правее обнулятся (так как они различны), а все разряды левее останутся как в исходном числе. Например,

140010 = 1 0 1 0 1 1 1 1 0 0 0 2

&

139910 = 1 0 1 0 1 1 1 0 1 1 1 2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

= 1 0 1 0 1 1 1 0 0 0 0 2

Таким образом, количество единиц в исходном числе уменьшится на 1.

Если выполнять данную последовательность операций (уменьшить на единицу и выполнить операцию побитового И) до тех пор пока в числе имеются единичные разряды, то количество таких операций будет равно числу единичных разрядов.

#include <stdlib.h>

#include <iostream>

using namespace std;

class Alpha // класс множества латинских букв

{

private:

unsigned bin; // бинарный код подмножества латинских букв

public:

Alpha () { bin=0; }; // конструктор по умолчанию

Alpha (char\*); // конструктор инициализации по стороке

int operator, ( Alpha& ); // перегразка оператора ,

int operator () ( Alpha&, Alpha& ); // перегрузка оператора ( )

int pop ( unsigned ); // быстрый подсчет единичных разрядов

friend ostream& operator << (ostream&, Alpha& ); // перегрузка оператора <<

operator char\*(); // перегрузка оператора приведение типа

};

// конструктор множества букв по строке

Alpha::Alpha ( char\* s )

{

bin=0;

while(\*s)

{

bin |= (1 << (tolower(\*s)-'a'));

s++;

}

}

// оператор вывода подмножеств букв

ostream& operator << (ostream& out, Alpha& z)

{

unsigned bit=1;

int i;

for (i=0; i<26; i++)

{

if((z.bin & bit) > 0)

out << (char)('a'+i);

bit = bit << 1;

}

return out;

}

//оператор преобразования множества в строку

Alpha::operator char\* ()

{

static char s[32];

unsigned b=bin;

int i=0;

int j=0;

while(b>0)

{

if (b & 1)

s[j++] = 'A' + i;

i++;

b >>= 1;

}

s[j] = 0;

return (s);

}

// скалярное произведение

int Alpha::operator , (Alpha& y)

{

return pop( bin & y.bin); // ^ - оператор побитового И

// pop – функция быстрого подсчета ед. разрядов

}

// расстояние Хемминга

int Alpha::operator () (Alpha& x, Alpha& y)

{

return pop( x.bin ^ y.bin ); // ^ - оператор побитового исключающего ИЛИ

// pop – функция быстрого подсчета ед. разрядов

}

// быстрый подсчет единичных разрядов

int Alpha::pop(unsigned b) // в функцию передается беззнаковое число

{

int i=0; // счетчик единичных разрядов

while ( b != 0) // пока в числе имеются 1 разряды

{

b = b & ( b - 1); // число уменьшается на 1 и применяется &

i++; // счетчик разрядов увеличивается на 1

}

return ( i ); // возврат числа единичных разрядов

}

//основная функция

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

{

Alpha A(argv[1]); // буквы 1-го аргумента

Alpha B(argv[2]); // буквы 2-го аргумента

Alpha H; // пустой набор – интерфейсный объект

// операции вычисления расстояния Хемминга

int d = H(A, B); // вычисление расстояния Хемминга

// =H.operator () (A, B);

int s = (A, B); // вычисление скалярного произведения

// =A.operator , ( B );

cout << "H (" << A << ", " << B << " ) = " << d << "\n";

// красным выделен перегруженный

// в классе оператор <<

cout << "(" << ( char\* ) A; // красным выделен перегруженный

cout << ", " << ( char\* ) B; // в классе оператор приведения типа

cout << " ) = " << s << "\n";

return (0);

}