Олимпиада школьников «Шаг в будущее» по общеобразовательному предмету "Информатика". 2014 год. Отборочный этап. 10-11 классы. Билет 1.

Задача 1: Ряд Фибоначчи (10)

Последовательность чисел 1, 1, 2, 3, 5, 8, 13, 21, …, a_n, …, два первых члена которой равны 1, а каждый член, начиная с третьего, равен сумме двух предыдущих: $a_{n+2} = a_{n+1} + a_n$, называется «рядом» Фибоначчи. Вычислить $a_n^4 - a_{n-2}a_{n-1}a_{n+1}a_{n+2}$.

Входные данные. Во входном файле записано одно целое число n ($3 \le n \le 10^{18}$).
Выходные данные. В выходной файл вывести одно целое число – значение вычисленного выражения.

Пример входного файла	Пример выходного файла
8	1

РЕШЕНИЕ:

```
#include "stdafx.h"

FILE *ifs, *ofs;

int _tmain(int argc, _TCHAR* argv[])
{
	unsigned long long n;
	ifs = fopen( "1.IN", "r" );
	fscanf( ifs, "%lld", &n );
	fclose( ifs );
	ofs = fopen( "1.OUT", "w" );
	fprintf( ofs, "%d\n", 1 ); // всегда 1
	fclose( ofs );
	return 0;
}
```

Задача 2: Сумма чисел (15)

Разобьем ряд натуральных чисел в группы: 1, (2, 4), (3, 5, 7), (6, 8, 10, 12), (9, 11, 13, 15, 17), … . Найти сумму чисел n-й группы.

Входные данные. Во входном файле записано одно целое число n – номер группы ($1 \le n \le 10^6$).
Выходные данные. В выходной файл вывести одно целое число – значение вычисленной суммы.

Примеры входного файла	Примеры выходного файла
10	510
100	500100

РЕШЕНИЕ:

```
#include "stdafx.h"

FILE *ifs, *ofs;

int _tmain(int argc, _TCHAR* argv[])
{
	unsigned long long n;
	ifs = fopen( "2.IN", "r" );
	fscanf( ifs, "%lld", &n );
	fclose( ifs );
	ofs = fopen( "2.OUT", "w" );
// Аналитическое решение
	unsigned long long m;
	if (n % 2 == 0) m = 2;
	else m = 1;
	fprintf( ofs, "%lld\n", n*(n*n + m)/2 );
// Алгоритмическое решение
	unsigned long long k, l, s, i, j;
	if (n == 1)
		s = 1;
```

```
	else if (n == 2)
		s = 6;
	else
	{
		k = (n-1)/2;
		if (n % 2 == 0)
		{
			m = k*(k+1);
			l = 2 + 2*(m-1);
		}
		else
		{
			m = k*k;
			l = 1 + 2*(m-1);
		}
		s = 0;
		for (i=1, j=l+2; i<n+1; i++, j+=2) s+=j;
	}
	fprintf( ofs, "%lld\n", s );
	fclose( ofs );
	return 0;
}
```

Задача 3: Текст (20)

Дан текст на английском языке. Предложения в тексте начинаются с заглавной буквы, а в конце предложения ставится "." (точка). Слова в предложениях и сами предложения отделяются друг от друга, по крайней мере, одним пробелом. Между словами допускаются символы пунктуации: "," (запятая), ";" (точка с запятой), ":" (двоеточие), "–" (тире). Символ апострофа считается частью слова. Определить, сколько в исходном тексте имеется слов, состоящих из одного, двух, трех и т.д. символов.

Входные данные. Во входном файле записан текст на английском языке. Длина строки текста не более 60. Количество строк текста не более 1000. Длина слова не более 20.
Выходные данные. Каждую пару чисел (количество символов и количество слов) печатать с новой строки.

Пример входного файла	Пример выходного файла
I'll tell you a story, a story anon, Of a noble prince, and his name was King John. For he was a prince, and a prince of great might, He held up great wrongs, he put down great right.	1 5 2 6 3 8 4 8 5 8 6 4

РЕШЕНИЕ:

```
#include "stdafx.h"
#include <string.h>

FILE *ifs, *ofs;

int _tmain(int argc, _TCHAR* argv[])
{
	char line[100], *token;
	char seps[] = " .,;:-\n\r";
	int counts[21] =  { 0 };
	int i, length;
	int high = 20;
	ifs = fopen( "3.IN", "r");
	ofs = fopen( "3.OUT", "w");
	while (!feof( ifs ))
	{
		if (fgets( line, 100, ifs ) != NULL)
		{
			token = strtok( line, seps );
			while (token != NULL)
			{
				/*fprintf( ofs, "%s\n", token );*/
```

```
                        length = strlen( token );
                        if (length > 0 && length < 21) counts[length]++;
                        token = strtok( NULL, seps );
                  }
            }
      }
      for (i = 20; i > 0; i--)
            if (counts[i] != 0) break;
            else high--;
      if (high != 0)
            for (i = 1; i <= high; i++)
                  fprintf( ofs, "%d\t%d\n", i, counts[i] );
      fclose( ifs );
      fclose( ofs );
      return 0;
}
```

Задача 4: Выражение (25)

Обычный метод записи математических выражений, в которых бинарный оператор записывается между операндами, известен под названием инфиксной записи. При отсутствии скобок операции выполняются согласно правилам приоритета операторов. Для изменения порядка выполнения операций применяют скобки. Префиксная польская запись – это форма записи математических выражений, в которой оператор располагается слева от операндов. Если оператор имеет фиксированную арность, то в такой записи будут отсутствовать скобки, и она может быть интерпретирована без неоднозначности. Для заданного инфиксного логического выражения получить последовательность операторов в префиксной польской записи.

Входные данные. Входной файл содержит одну строку, в которой записано инфиксное логическое выражение. В исходном выражении нет пробелов, в качестве операндов используются латинские буквы в верхнем или нижнем регистре, в качестве операторов используются знаки "~" (отрицание), "&" (конъюнкция), "|" (дизъюнкция). Длина каждого выражения не превосходит 100.
Выходные данные. В выходной файл вывести одну строку, в которой записаны операторы префиксной польской записи.

Пример входного файла	Пример выходного файла
x&~(x&~y)&(z|y)	&&~&~|

РЕШЕНИЕ:

```
#include "stdafx.h"
#include <ctype.h>
#include <string.h>

const int N = 100;

typedef struct btree BTREE;

struct btree {
      char info;
      BTREE *left;
      BTREE *right;
};

FILE *ifs, *ofs;

static int op_stack[N+1];
static int ex_stack[N+1];
static int op_index = 0, ex_index = 0;
BTREE *root = 0;

void op_push( int value )
{
      op_stack[++op_index] = value;
}

int op_pop()
{
      op_index--;
```

```
	return (op_stack[op_index+1]);
}

int op_top()
{
	return (op_stack[op_index]);
}

void ex_push( int value )
{
	ex_stack[++ex_index] = value;
}

int ex_pop()
{
	ex_index--;
	return (ex_stack[ex_index+1]);
}

int ex_top()
{
	return (ex_stack[ex_index]);
}

int is_operator( int c )
{
	if (c == '|' || c == '&' || c == '~') return 1;
	else return 0;
}

int op_priority( int op )
{
	if (op == '~') return 4;
	if (op == '&') return 3;
	if (op == '|') return 2;
	return 1;
}

void printtree( BTREE *root )
{
	if (!isalpha( root->info )) fprintf( ofs, "%c", root->info ); // Печатаем текущий
узел
	if (root->left) printtree( root->left ); // Обходим левое поддерево
	if (root->right) printtree( root->right ); // Обходим правое поддерево
}

void buildtree( BTREE *&root )
{
	char top;
	if (ex_index == 0) return;
	top = ex_top();
	if (root == 0)
	{
		root = new BTREE;
		root->info = top;
		root->left = 0;
		root->right = 0;
	}
	if (is_operator( top ))
	{
		if (top == '~')
		{
			ex_pop();
			buildtree( root->right );
		}
		else
		{
```

```
                ex_pop();
                buildtree( root->right );
                buildtree( root->left );
            }
        }
        else ex_pop();
}

int _tmain(int argc, _TCHAR* argv[])
{
        char line[N+1];
        int m, i;
        char ch, top;
        // Читаем строку выражения
        ifs = fopen( "4.IN", "r" );
        fgets( line, N, ifs );
        fclose( ifs );
        m = strlen( line );
        if (m == 0) return 1;
        // Реализуем алгоритм сортировочной станции
        op_stack[0] = 0;
        ex_stack[0] = 0;
        for (i = 0; i < m; i++)
        {
            ch = line[i];
            if (isalpha( ch )) ex_push( ch );
            else if (ch == '(') op_push( ch );
            else if (ch == ')')
            {
                top = op_top();
                while (top != '(' && op_index != 0)
                {
                    ex_push( top );
                    op_pop();
                    top = op_top();
                }
                if (top == '(') op_pop();
            }
            else if (is_operator( ch ))
            {
                top = op_top();
                while (is_operator( top ) && op_priority( top ) >= op_priority( ch )
&& op_index != 0)
                {
                    ex_push( top );
                    op_pop();
                    top = op_top();
                }
                op_push( ch );
            }
            else;
        }
        top = op_top();
        while (is_operator( top ) && op_index != 0)
        {
            ex_push( top );
            op_pop();
            top = op_top();
        }
        buildtree( root ); // Строим дерево
        ofs = fopen( "4.OUT", "w" );
        printtree( root ); // Обходим дерево
        fprintf( ofs, "\n" );
        fclose( ofs );
        return 0;
}
```

Задача 5: Путь в многограннике (30)

Рассматривается 3-мерный выпуклый многогранник. Требуется перейти с начальной грани на конечную грань, пересекая наименьшее число ребер. Пересекать ребро в вершине многогранника нельзя.

Входные данные. В первой строке входного файла содержится целое число N – количество граней ($4 \le N \le 6000$). В последующих N строках записаны целые числа m, x_1, y_1, z_1, …, x_m, y_m, z_m, где m – количество сторон грани ($3 \le m \le 20$), а x_i, y_i, z_i – координаты вершин грани и ($-10^6 \le x_i, y_i, z_i \le 10^6$) для всех i. В последней строке входного файла записаны два целых числа: s – номер начальной грани и t – номер конечной грани (нумерация граней начинается с 1).
Выходные данные. В выходной файл вывести одно целое число – наименьшее количество пересеченных ребер.

Пример входного файла	Пример выходного файла
6 4 0 0 0 0 0 1 0 1 1 0 1 0 4 1 0 0 1 0 1 1 1 1 1 1 0 4 0 0 0 1 0 0 1 0 1 0 0 1 4 0 1 0 1 1 0 1 1 1 0 1 1 4 0 0 0 0 1 0 1 1 0 1 0 0 4 0 0 1 0 1 1 1 1 1 1 0 1 1 2	2

РЕШЕНИЕ:

```
#include "stdafx.h"
#include <iostream>
#include <vector>
#include <queue>
#include <map>

using namespace std;

struct Edge
{
	int x1, y1, z1, x2, y2, z2;
	Edge( int a1, int b1, int c1, int a2, int b2, int c2 )
	{
		if ((a1 < a2) || (a1 == a2 && b1 < b2) || (a1 == a2 && b1 == b2 && c1 <=
c2))
		{
			x1 = a1;
			y1 = b1;
			z1 = c1;
			x2 = a2;
			y2 = b2;
			z2 = c2;
		}
		else
		{
			x1 = a2;
			y1 = b2;
			z1 = c2;
			x2 = a1;
			y2 = b1;
			z2 = c1;
		}
	}

	bool operator<( const Edge &e ) const
	{
		if (x1 != e.x1) return x1 < e.x1;
		if (y1 != e.y1) return y1 < e.y1;
		if (z1 != e.z1) return z1 < e.z1;
		if (x2 != e.x2) return x2 < e.x2;
		if (y2 != e.y2) return y2 < e.y2;
		if (z2 != e.z2) return z2 < e.z2;
		return false;
	}
```

```
};

int _tmain(int argc, _TCHAR* argv[])
{
	int n, m, q, x, y, x1, y1, z1, x2, y2, z2, x3, y3, z3;
	cin >> n;
	while (n)
	{
		vector< vector<int> > nodes(n);
		map<Edge, int> edges;
		for(int i = 0; i < n; i++)
		{
			cin >> m >> x1 >> y1 >> z1;
			x3 = x1;
			y3 = y1;
			z3 = z1;
			for(int k = 0; k < m; k++)
			{
				if (k < m - 1)
					cin >> x2 >> y2 >> z2;
				else
				{
					x2 = x3;
					y2 = y3;
					z2 = z3;
				}
				Edge e( x1, y1, z1, x2, y2, z2 );
				if (edges.find(e) != edges.end())
				{
					int j = edges[e];
					nodes[i].push_back(j);
					nodes[j].push_back(i);
				}
				else
					edges[e] = i;
				x1 = x2;
				y1 = y2;
				z1 = z2;
			}
		}

		for(int i=0;i<n;i++) {
			cout << i << ": ";
			for(int j=0;j<nodes[i].size();j++)
				cout << nodes[i][j] << " ";
			cout << endl;
		}

		cin >> q;
		while (q--)
		{
			cin >> x >> y;
			x--;
			y--;
			vector<int> vis(n,-1);
			queue<int> cur;
			vis[x] = 0;
			cur.push(x);
			while (vis[y] == -1)
			{
				x = cur.front();
				cur.pop();
				for(int i = 0; i < nodes[x].size(); i++)
				{
					int z = nodes[x][i];
					if (vis[z] == -1)
					{
```

```
                                    vis[z] = vis[x]+1;
                                    cur.push(z);
                                }
                            }
                        }
                        cout << vis[y] << endl;
                }
                cin >> n;
        }
        return 0;
}
```