Лабораторная работа №2. Работа с внешними файлами, использование массивов и операций манипулирования битами.

Настройки среды Microsoft Visual Studio

Пути поиска стандартных заголовочных файлов

Вы уже знаете, что при помощи директивы препроцессора #include можно подключать заголовочные файлы внешних библиотек. Например, для подключения заголовочного файла **stdio.h**, содержащего функции ввода-вывода стандартной библиотеки языка Си, необходимо использовать директиву:

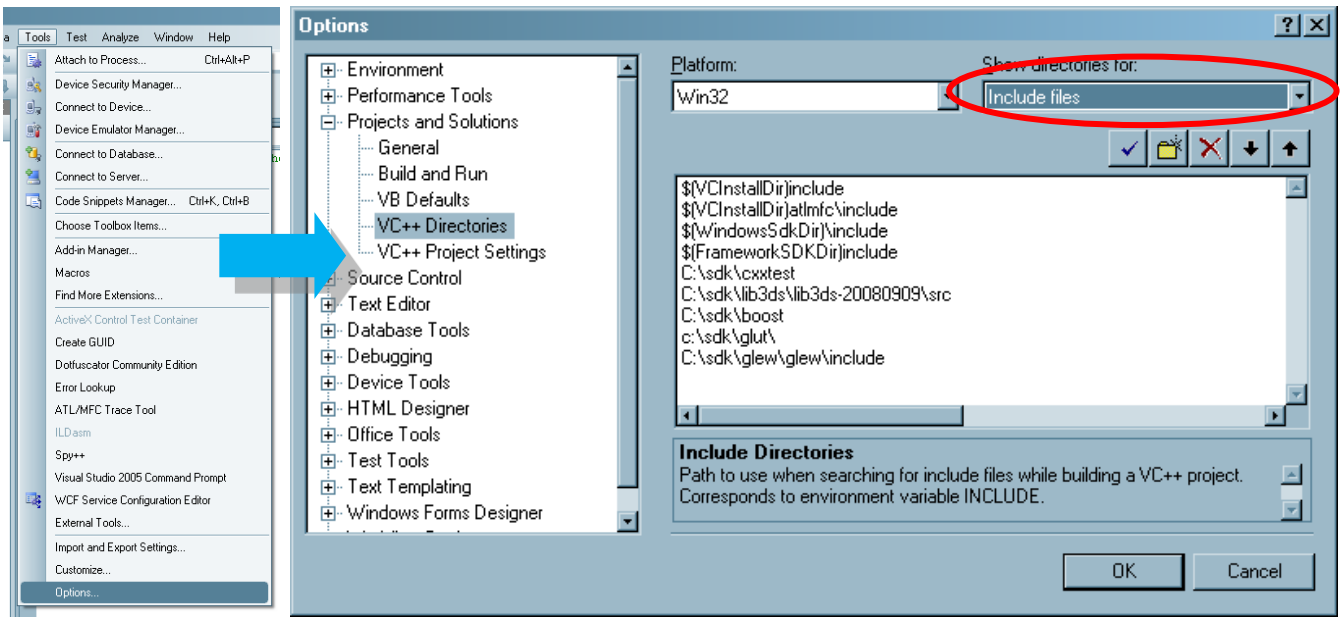
```
#include <stdio.h>
```

Если имя файла в директиве **#include** заключено **в угловые скобки**, то поиск указанного файла производится сначала в директориях со стандартными заголовочными файлами, а затем относительно текущего файла.

Если имя файла в директиве **#include** заключено **в двойные кавычки**, то поиск производится сначала относительно текущего файла, а затем в директориях со стандартными заголовочными файлами.

При этом подключен будет первый найденный заголовочный файл с указанным именем. Таким образом, для подключения стандартных заголовочных файлов следует использовать директиву **#include <имя файла>**, а для подключения заголовочных файлов Вашего проекта или решения – директиву **#include "имя файла"**.

Задать пути к директориям стандартных файлов в среде Microsoft Visual Studio 8 можно при помощи меню Tools→Options→Projects and Solutions→VC++ Directories.



На странице свойств необходимо выбрать желаемую платформу и тип файлов:

- **Executable files**. В данном списке указываются директории, в которых компилятор будет производить поиск исполняемых файлов, например, файлов вспомогательных утилит.
- **Include files**. В данном списке директорий будет производиться поиск файлов, подключаемых при помощи директивы #include. В нем следует размещать пути к директориям со стандартными заголовочными файлами внешних библиотек.
- **Reference files**. В данных директориях будет производиться поиск файлов со сборками и модулями платформы .NET.
- **Library files**. В данных директориях будет производиться поиск файлов внешних библиотек.
- **Source files**. В данных директориях будет производиться поиск исходных файлов, используемых технологией IntelliSense.
- **Exclude directories**. Пути к директориям, файлы в которых будут пропущены при выявлении зависимостей между файлами.

Вам может понадобиться вносить изменения в данные списки директорий, как правило, при установке библиотек третьих лиц, в основном, в разделы **Include files** и **Library files**. Вносить изменения в раздел **Executable files** может понадобиться в случае необходимости использования дополнительных утилит в процессе сборки проекта.

Функции работы с файлами

Стандартная библиотека языка Си предоставляет набор функций для файлового ввода-вывода. Впоследствии при ознакомлении со стандартной библиотекой языка Си++ вы познакомитесь с более надежными и продвинутыми способами работы с файлами.

Открытие файла

Для начала работы с файлом необходимо открыть его для чтения или записи. Сделать это можно при помощи функции **fopen**. Необходимо указать имя открываемого файла и режим работы с ним (для чтения или записи, текстовый или бинарный файл). В случае успешного открытия файла функция вернет ненулевой указатель.

```
FILE *pFile = fopen("file1.txt", "rb");
if (pFile == NULL)
{
    printf("File opening error\n");
    return 1;
}
```

> Отличие текстовых файлов от бинарных заключается в том, что в системе Windows символ \n (символ с кодом 10 в кодировке ASCII), выводимый в выходной **текстовый** файл будет записан в виде 2 байтов \r\n (символы с кодами 13 и 10 в кодировке ASCII). При чтении из файла, напротив, последовательности символов \r\n будет считана в виде одного символа \n.
>
> В случае с **бинарными** (двоичными) файлами никакой замены символов не происходит, и данные считываются именно в том виде, в каком они представлены в файле, а записываются в файл в том виде, в котором они хранились в памяти.

Закрытие файла

После того, как работа с файлом в программе закончена, **файл необходимо закрыть**. Несоблюдение этого правила может привести к тому, что данный файл окажется заблокированным для других программ до окончания работы Вашей программы. Кроме того, ресурсы компьютера, занимаемые открытым файлом, будут расходоваться впустую. А теперь представьте, что программа в ходе своей работы обрабатывает сотни или даже тысячи файлов и «забывает» (не без помощи программиста) их закрыть. В долгосрочной

перспективе это приведет к тому, что доступное программе количество ресурсов операционной системы будет исчерпано, и программа аварийно завершит свою работу.

Для закрытия файла, открытого при помощи функции **fopen** в стандартной библиотеке языка Си используется функция **fclose**.

```
if (pFile != NULL)
{
    // Указатель, передаваемый функции fclose, не должен быть равен NULL
    fclose(pFile);
    pFile = NULL;
}
```

Следует внимательно следить за тем, чтобы открытые файлы закрывались при всех путях выполнения программы. Особенное внимание следует обращать на циклы, ветвления, а также функции, имеющие несколько точек выхода.

> Язык C++ предоставляет гораздо более надежные средства освобождения ресурсов, занимаемых объектами программы. С ними Вы познакомитесь в одной из следующих лабораторных работ.

Чтение файла

Прочитать открытый для чтения файл можно при помощи функции fopen, можно прочитать при помощи следующих функций:

- fgetc для чтения одиночных символов
- fread для чтения блоков данных

Пример посимвольного чтения файла:

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    FILE *pFile = fopen("file1.txt", "rb");
    if (pFile == NULL)
    {
        printf("File opening error\n");
        return 1;
    }

    // Переменная ch должна иметь тип int, а не char
    int ch;
    // Цикл чтения продолжается пока код считанного символа не станет равен EOF (-1)
    while ((ch = fgetc(pFile)) != EOF)
    {
        putchar(ch);
    }

    fclose(pFile);
    pFile = NULL;

    return 0;
}
```

> Обратите внимание на то, что переменная ch объявлена в программе с типом **int**, а не **char**. Дело в том, что функция **fgetc** возвращает результат типа **int**, диапазон которого позволяет представлять не только символы с кодами от 0 до 255 (диапазон типа **unsigned char**), но еще и специальное значение EOF равное -1, которое зарезервировано для представления маркера конца файла.

Проблема с использованием типа **char** для хранения и обработки значений, возвращаемых функцией fgetc (а также функции getchar), заключается в том, что он позволяет представлять числа диапазона -128…+127, а функция fgetc возвращает значения от -1 до 255. Числа диапазона от 128 до 255 в переменной типа **char** в двоичном виде совпадают с числами диапазона от -128 до -1, причем 255 и -1 будут иметь одинаковое двоичное представление. В результате, программа будет ошибочно воспринимать символ с кодом 255 как маркер конца файла и наоборот, что в нашем случае вызвало бы прекращение цикла чтения файла при встрече символа с кодом 255 (в кодировке Windows-1251 это строчная буква «я»).

Пример блочного чтения данных из файла:

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    FILE *pFile = fopen("file1.txt", "rb");
    if (pFile == NULL)
    {
        printf("File opening error\n");
        return 1;
    }

    double arrayOfDouble[10];
    int readItemsCount = fread(&arrayOfDouble[0], sizeof(double), 10, pFile);
    printf("%d items were read\n", readItemsCount);

    fclose(pFile);
    pFile = NULL;

    return 0;
}
```

Запись файла

Для записи файла, открытого при помощи функции fopen можно воспользоваться следующими функциями стандартной библиотеки языка Си:

- fputc для посимвольного вывода данных в файл
- fwrite для блочного вывода данных в файл

Пример программы, выполняющей посимвольную запись в файл.

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    FILE *pFile = fopen("file1.txt", "wt");
    if (pFile == NULL)
    {
        printf("File opening error\n");
        return 1;
    }

    int ch;
    while ((ch = getchar()) != EOF)
    {
        fputc(ch, pFile);
    }

    fclose(pFile);
    pFile = NULL;
```

```
    return 0;
}
```

Пример программы, использующей блочную запись данных в файл.

```
#include <stdio.h>
#include <string.h>

int main(int argc, char* argv[])
{
    FILE *pFile = fopen("file1.txt", "wb");
    if (pFile == NULL)
    {
        printf("File opening error\n");
        return 1;
    }

    const char message[] = "Hello, world";
    fwrite(message, sizeof(char), strlen(message), pFile);

    fclose(pFile);
    pFile = NULL;

    return 0;
}
```

С другими функциями стандартной библиотеки языка Си, работающими с файлами, Вы можете познакомиться в Интернет, например, в MSDN.

Операторы манипулирования битами

Язык C++ предоставляет ряд операторов, выполняющих операции над отдельными битами своих аргументов. Аргументами данных операторов могут выступать только **целочисленные типы данных**.

Побитовые операции

Данные операторы позволяют осуществлять операции над отдельными битами целочисленных операндов.

- Оператор & (AND). Выполняет операцию «И» над соответствующими битами своих операндов.
- Оператор | (OR). Выполняет операцию «Или» над соответствующими битами своих операндов.
- Оператор ^ (XOR). Выполняет операцию «Исключающее ИЛИ» над соответствующими битами своих операндов.
- Оператор ~. Выполняет инвертирование битов (установленные в 1 биты сбрасываются в 0 и наоборот).
- Операторы << и >>, выполняющие сдвиг битов влево и вправо соответственно.

Операторы &, | и ^ являются бинарными (имеют 2 операнда) инфиксными (размещаются между своими операндами) операторами. Таблица истинности для данных операндов представлена ниже.

X	Y	X & Y	X | Y	X ^ Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Оператор ~ является унарным (имеет один операнд) префиксным (располагается перед своим операндом) оператором. Его таблица истинности представлена ниже.

X	~X
0	1
1	0

Операторы сдвига << и >> являются бинарными инфиксными операторами и выполняют сдвиг битов аргумента влево или вправо на заданное количество разрядов. Синтаксис операторов:

«аргумент» << «*количество разрядов»*
и
«аргумент» >> «*количество разрядов»*

Оператор сдвига влево выполняет сдвиг битов влево, заполняя освободившиеся позиции нулями.
Например, значение выражения
5 << 3
равно
$00000101_2 << 3 = 00101000_2 = 40$

Со сдвигом вправо ситуация обстоит гораздо интереснее, т.к. результат работы данного оператора зависит от того, является ли «*аргумент*» целым числом со знаком или без знака.

В том случае, когда аргумент является числом **без знака**, то освободившиеся при сдвиге биты заполняются **нулями**, как и в случае со сдвигом вправо, а сам результат будет также беззнаковым. Например, в результате работы следующая программы

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    unsigned char value = 0xf0; // 0xf0 = 11110000₂
    printf("%d", value >> 2);   // 00111100₂ = 60

    return 0;
}
```

в стандартный поток вывода будет выведено число 60.

В случае, когда аргумент является числом со знаком, то результат будет также числом со знаком, а освободившиеся при сдвиге биты заполняются значением старшего значащего бита аргумента. Следующая программа

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    signed char value = 0xf0;  // 0xf0 = 11110000₂
    printf("%d", value >> 2);  // 11111100₂ = -4

    return 0;
}
```

выведет в стандартный поток вывода число -4.

Бинарные операторы манипулирования битами могут быть скомбинированы с оператором = и образовывать следующие операторы:
&=, |=, ^=, <<= и >>=, выполняющие присваивание после выполнения соответствующих побитовых операций. Пример:

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    int a = 5;
    a &= 3;
    printf("%d", a);

    return 0;
}
```

В стандартный поток вывода будет выведено число 1 (5 & 3 = 1).

Практические задания

На оценку «**удовлетворительно**» необходимо выполнить **все обязательные задания** и набрать не менее 80 баллов

На оценку «**хорошо**» необходимо выполнить **все обязательные задания** и часть дополнительных и набрать не менее 250 баллов

На оценку «**отлично**» необходимо выполнить все обязательные задания и часть дополнительных и набрать не менее 400 баллов

Внимание, дополнительные задания принимаются только **после успешной защиты** обязательных заданий.

Обязательные задания

Задание 1

Выполните задание одного из предложенных вариантов.

Вариант №1 – 20 баллов

Разработайте программу **copyfile.exe**, выполняющую копирование одного **текстового** файла в другой. Параметры командной строки:

```
copyfile.exe <input file name> <output file name>
```

Программа должна корректно обрабатывать ошибки, связанные с файловыми операциями, а также корректно закрываться открываемые программой файлы.

В комплекте с программой должны обязательно поставляться файлы, позволяющие проверить ее работу в автоматическом режиме.

Вариант №2 – 40 баллов

Разработайте программу **compare.exe**, выполняющую сравнение содержимого **текстовых** файлов. Формат командной строки:

```
compare.exe <file1> <file2>
```

В том случае, если файлы имеют одинаковое содержимое, программа должна вернуть нулевой результат в операционную систему и вывести строку «Files are equal” в стандартный поток вывода. В противном случае программа должна вывести номер первой строки, в которой были найдены расхождения в содержимом файлов: «Files are different. Line number is <номер строки>», и вернуть значение 1.

Программа должна корректно обрабатывать ошибки, связанные с файловыми операциями.

В комплекте с программой должны обязательно поставляться файлы, позволяющие проверить корректность её работы в автоматическом режиме.

Вариант №3 – 50 баллов

Разработайте программу **findtext.exe**, выполняющую поиск указанной строки в файле. Формат командной строки:

```
findtext.exe <file name> <text to search>
```

Например:

```
findtext.exe “Евгений Онегин.txt” “Я к Вам пишу”
```

В случае, когда искомая строка в файле найдена, приложение возвращает **нулевое** значение и выводит в стандартный выводной поток номера всех строк (по одному номеру в каждой строке), содержащих искомую строку. В противном случае программа возвращает 1 и выводит в стандартный поток вывода «Text not found».

При осуществлении поиска регистр символов имеет значение (это упрощает поиск). Слова «**Онегин**» и «**онегин**» являются разными.

Программа должна корректно обрабатывать ошибки, связанные с файловыми операциями.

В комплекте с программой должны обязательно поставляться файлы, позволяющие проверить корректность её работы в автоматическом режиме.

Задание 2

Выполните задание одного из предложенных вариантов

Вариант №1 – 20 баллов

Разработайте приложение **calcbits.exe**, выполняющее подсчет и вывод в output количества единичных битов в байте. Для подсчета единичных битов используйте **операции манипулирования битами**. Формат командной строки:

```
calcbits.exe <byte>
```

где **byte** – число в десятичное системе, в двоичном представлении которого должен производиться подсчет установленных в единицу бит.

Например, вызов

```
calcbits.exe 5
```

должен выводить в output значение 2 (число 5 в двоичной системе равно 00000101).

В случае некорректных входных данных программа должна выводить пользователю сообщение об ошибке и возвращать ненулевое значение.

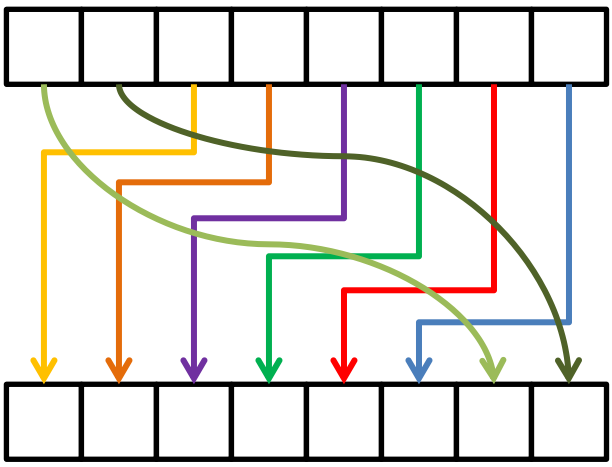
В комплекте с программой должны обязательно поставляться файлы, позволяющие проверить ее работу в автоматическом режиме.

Вариант №2 – 30 баллов

Разработайте приложение **rotatebyte.exe**, выполняющее **циклическое вращение** битов в байте влево или вправо на заданное количество разрядов. Для решения задачи используйте операции манипулирования битами. Формат командной строки:

```
rotatebyte.exe <byte> <number of bits> <L / R>
```

На следующей диаграмме показан пример циклического сдвига некоторого байта на 2 разряда влево.



Например, в результате вызова

```
rotatebyte.exe 17 2 R
```

в стандартный поток вывода программа должна вывести **68**

В случае некорректных входных данных (например, если вместо числа программе было передано что-то другое) программа должна выводить пользователю сообщение об ошибке и возвращать ненулевое значение.

В комплекте с программой должны обязательно поставляться файлы, позволяющие проверить ее работу в автоматическом режиме.

Вариант №3 – 30 баллов

Разработайте программу **bin2dec.exe**, выполняющую перевод числа из двоичной системы в десятичную и вывод результата в стандартный поток вывода. Формат командной строки:

```
bin2dec.exe <число в двоичной системе>
```

Входной параметр задается в виде целого числа без знака, заданного в двоичной системе в диапазоне от 0 до 2^{32-1}.

Например, в результате вызова

```
bin2dec.exe 110010000
```

в стандартный поток вывода должно быть выведено число 400.

Для перевода числа из двоичной системы в десятичную систему используйте арифметические операции и операции манипулирования битами.

В том случае, когда в качестве входного параметра указано некорректное двоичное число, программа должна вывести соответствующее сообщение об ошибке.

В комплекте с программой должны обязательно поставляться файлы, позволяющие проверить ее работу в автоматическом режиме.

Вариант №4 – 30 баллов

Разработайте программу **dec2bin.exe**, выполняющую перевод числа из десятичной системы в двоичную и вывод результата в стандартный поток вывода. Формат командной строки:

```
bin2dec.exe <число в десятичной системе>
```

Входной параметр задается в виде целого числа без знака , заданного в двоичной системе в диапазоне от 0 до 2^{32-1}. Выводимое число в двоичной не должно содержать незначащих нулевых бит.

Например, в результате вызова

```
bin2dec.exe 253
```

в стандартный поток вывода должно быть выведено число **11111101**.

Для перевода числа из десятичной системы в двоичную систему используйте арифметические операции и операции манипулирования битами.

В том случае, когда в качестве входного параметра указано некорректное десятичное число, программа должна вывести соответствующее сообщение об ошибке.

В комплекте с программой должны обязательно поставляться файлы, позволяющие проверить ее работу в автоматическом режиме.

Вариант №5 – 50 баллов

Разработайте приложение **flipbyte.exe**, выполняющее изменение порядка следования двоичных битов в 8-битовом целом числе (байте), заданном в десятичном представлении, на противоположный. Для этого используйте операторы для работы с битами. «Перевернутый» байт выводится в output также в десятичном представлении с завершающим символом перевода строки \n. Формат командной строки приложения:

```
flipbyte.exe <входной байт>
```

Например, в результате вызова

```
flipbyte.exe 6
```

в output должно быть выведено 96 ($6_{10} = 00000110_2$, после изменения порядка битов данное число превратится в $01100000_2 = 96_{10}$)

Некорректные входные данные (например, передача строки, не являющейся десятичным числом, или числа, выходящего за пределы 0-255) должно выводиться соответствующе сообщение об ошибке.

В комплекте с программой должны обязательно поставляться файлы, позволяющие проверить ее работу в автоматическом режиме.

Вариант №6 – 50 баллов

Разработайте приложение **solve.exe**, выполняющее нахождение корней квадратного уравнения, поступающего через командную строку в виде коэффициентов A, B и C квадратного уравнения

$A\mathbf{x}^2 + B\mathbf{x} + C = 0$

Формат командной строки:

```
solve.exe <A> <B> <C>
```

Корни квадратного уравнения выводятся разделенные пробелами с точностью **до 4 знаков после запятой**. В случае отсутствия корней (дискриминант меньше нуля) выводится результат **There is no real root**. В случае одного единственного корня (дискриминант равен нулю) выводится один единственный корень. В случае, когда параметр A равен 0, и уравнение уже не является квадратным, необходимо вывести соответствующее сообщение пользователю об ошибке.

В комплекте с программой должны обязательно поставляться файлы, позволяющие проверить ее работу в автоматическом режиме.

Задание 3

Выполните задание одного из предложенных вариантов

Вариант №1 – 40 баллов

Разработайте приложение **multmatrix.exe**, выполняющее перемножение двух матриц размером 3*3, коэффициенты которых заданы во входных файлах (смотрите **matrix.txt** в качестве иллюстрации), и выводящее результат умножения в стандартный поток вывода. Формат командной строки приложения:

```
multmatrix.exe <matrix file1> <matrix file2>
```

Коэффициенты матриц задаются в текстовых файлах в трех строках по 3 элемента.

Коэффициенты результирующей матрицы выводятся **с точностью до 3 знаков после запятой**.

Используйте двухмерные массивы для хранения коэффициентов матриц.

В комплекте с программой должны обязательно поставляться файлы, позволяющие проверить ее работу в автоматическом режиме.

Вариант №2 – 40 баллов

Разработайте приложение **invert.exe**, выполняющее инвертирование матрицы 3*3, т.е. нахождение обратной матрицы и выводящее коэффициенты результирующей матрицы в стандартный поток вывода. Формат командной строки приложения:

```
multmatrix.exe <matrix file1>
```

Коэффициенты входной матрицы заданы во входном текстовом файле (смотрите файл **matrix.txt** в качестве иллюстрации) в трех строках по 3 элемента.

Коэффициенты результирующей матрицы выводятся **с точностью до 3 знаков после запятой**.

Используйте двухмерные массивы для хранения коэффициентов матриц.

В комплекте с программой должны обязательно поставляться файлы, позволяющие проверить ее работу в автоматическом режиме.

Дополнительные задания

Задание 1

Выполните задания одного из предложенных вариантов.

Вариант №1 – 100 баллов

Разработайте приложение **rle.exe**, выполняющее RLE-компрессию бинарных файлов с сильно разреженным содержимым, а также декомпрессию упакованных ею файлов. Необходимо реализовать следующий принцип компрессии файла:

При обнаружении последовательности одинаковых байтов, она кодируется при помощи двух байтов. Первый байт хранит количество повторов следующего за ним байтов. Например, последовательность байт AAABBBBC, будет представлена в виде следующей **последовательности байт**:

3, 'A', 4, 'B', 1, 'C'

Таким образом, исходная 8-байтовая последовательность будет представлена при помощи всего шести байт.

Входные файлы нулевой длины представляются в виде выходных файлов нулевой длины.

Очевидно, что последовательность одинаковых байтов длиной более 255 байт двумя байтами закодирована быть не может, в этом случае первые 2 байта кодируют первые 255 байт последовательности, а затем обрабатываются следующие байты последовательности. Также очевидно, что однобайтовые последовательности требуют в 2 раза больше данных. Тем не менее, в качестве учебного задания для ознакомления с функциями файлового ввода-вывода задание вполне подходит.

Формат параметров командной строки.

Упаковка:

```
rle.exe pack <input file> <output file>
```

Распаковка:

```
rle.exe unpack <input file> <output file>
```

Программа должна корректно обрабатывать ошибки, связанные с открытием входных и выходных файлов.

Размеры входных и выходных файлов ограничены 2 Гб.

В комплекте с программой должен обязательно поставляться .bat файл, позволяющий проверить работу программы в автоматическом режиме, а также эталонные значения входных и выходных файлов. Проверка должна проверять как корректность работы в режиме упаковки, так и в режиме распаковки. Особое внимание следует уделить проверке работы граничных условий работы программы:

- Входной файл нулевой длины
- Тестирование файлов, содержащих последовательности одинаковых символов в 255, 256 и 257 байт
- Недопустимые ситуации: нечетная длина запакованного файла, количество повторений символа, равное нулю
- Файл, содержащий символы с кодом 255

Вариант №2 – 80 баллов

Разработайте программу **replace.exe**, выполняющую замену подстроки в текстовом файле на другую строку, и записывающей результат в выходной файл (отличный от входного). Формат командной строки:

```
replace.exe <input file> <output file> <search string> <replace string>
```

Программа должна корректно обрабатывать ошибки, связанные с открытием входных и выходных файлов. Искомая строка не может быт пустой.

Размеры входных и выходных файлов ограничены 2 Гб. Размеры искомой строки и строки-заменителя не ограничены.

Внимание, программа корректно должна обрабатывать ситуацию, когда длина искомой строки равна нулю. В этом случае замены символов производиться не должно.

В комплекте с программой должны обязательно поставляться файлы, позволяющие проверить ее работу в автоматическом режиме:

- .bat файл, выполняющий запуск программы с различными тестовыми параметрами. Необходимо проверить возможные граничные условия программы, включая недопустимые ситуации.
- Эталонные входные и выходные файлы для проверки работы программы с тестовыми входными данными.

Вариант №3 – 60 баллов

Разработайте программу **join.exe**, выполняющую слияние содержимого нескольких входных **бинарных файлов** в выходной файл (отличный от входных). Формат командной строки:

```
join.exe <input file1> … <input file N> <output file>
```

Программа должна корректно обрабатывать ошибки, связанные с открытием входных и выходных файлов. Размеры входных и выходных файлов ограничены 2 Гб

В комплекте с программой должны обязательно поставляться файлы, позволяющие проверить ее работу в автоматическом режиме.

Вариант №4 – 60 баллов

Разработайте приложение **bmpinfo.exe**, выполняющее считывание заголовка входного файла и, если, судя по заголовку, формат файла соответствует признакам формата BMP (информацию о структуре BMP файла можно найти в Интернет), то необходимо вывести в output следующую информацию:

- Разрешение (ширина и высота файла)
- Количество бит на пиксель
- В случае, если в BMP файле используется палитра (8 и менее бит), необходимо вывести количество используемых цветов.
- Размер изображения в байтах

Если входной файл не является файлом формата BMP, то вывести об этом соответствующее сообщение.

Формат командной строки:

```
bmpinfo.exe <input file name>
```

Программа должна корректно обрабатывать ошибки, связанные с файловыми операциями.

Размеры входных и выходных файлов ограничены 2 Гб

В комплекте с программой должны обязательно поставляться файлы, позволяющие проверить ее работу в автоматическом режиме.

Примечание: BMP-файлы (редко) могут использовать PNG, RLE или Jpeg-компрессию данных – в этом случае нужно вывести соответствующую информацию

Вариант №5 – 60 баллов

Разработать приложение **extract.exe**, выполняющее извлечение фрагмента произвольной длины входного **бинарного** файла начиная с произвольной позиции в выходной файл. Формат командной строки:

```
extract.exe <input file> <output file> <start position> <fragment size>
```

Программа должна корректно обрабатывать ошибки, связанные с открытием входных и выходных файлов, а также корректностью параметров start position и fragment size. В частности, выход за пределы исходного файла является недопустимым.

Размеры входных и выходных файлов ограничены 2 Гб

В комплекте с программой должны обязательно поставляться файлы, позволяющие проверить ее работу в автоматическом режиме.

Задание 2

Выполните задания одного из предложенных вариантов

Вариант №1 – 100 баллов

Разработайте программу **radix.exe**, выполняющую перевод чисел из одной произвольной системы счисления в другую произвольную и запись результата в стандартный поток вывода. Под произвольной системой счисления понимается система с основанием от 2 до 36. Системы счисления с 11-ричной до 36-ричной должны использовать **заглавные** буквы латинского алфавита от A до Z для представления разрядов с 11_{10} до 35_{10}. Формат командной строки приложения:

```
radix.exe <source notation> <destination notation> <value>
```

Например, следующим способом программа должна осуществлять перевод шестнадцатеричного числа 1F в его десятичное представление:

```
radix.exe 16 10 1F
```

В конце строки, выводимой в стандартный поток вывода **должен располагаться код \n.**

Программа должна быть способна осуществлять перевод **как положительных, так и отрицательных чисел**, а также нуля. Особое внимание уделите переводу максимальных и минимальных целых чисел на данной платформе (они должны преобразовываться корректно).

Программа должна корректно обрабатывать ошибки

Внимание, для перевода строкового представления в числовое и числового в строковое в произвольных системах счисления должны быть разработаны функции:

```
int StringToInt(const char str[], int radix, bool & wasError);
```

и

```
void IntToString(int n, int radix, char str[], int bufferLength, bool & wasError);
```

В случае ошибок (некорректные значения входных параметров, переполнение при переводе строки в число, нехватка места в строковом массиве для представления числа) данные функции **должны изменять состояние булевой переменной wasError**.

В случае обнаружения ошибки программа должна вывести соответствующее сообщение и корректно завершить свою работу.

В комплекте с программой должны обязательно поставляться файлы, позволяющие проверить ее работу в автоматическом режиме.

Вариант №2 – 150 баллов

Разработайте приложение **crypt.exe**, выполняющее шифрование/дешифрование содержимого входного бинарного файла, и записывающего результат в выходной файл. Формат командной строки:

для шифрования:

```
crypt.exe crypt <input file> <output file> <key>
```

для дешифрования

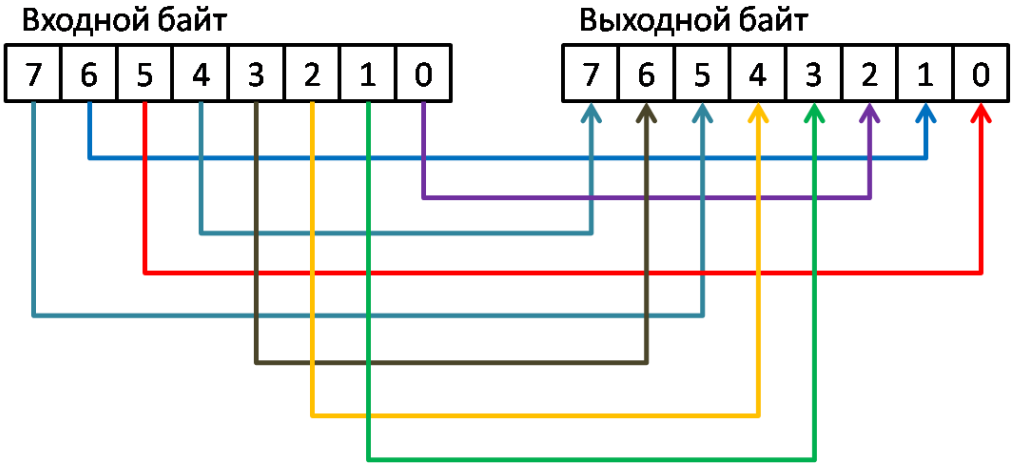
```
crypt.exe decrypt <input file> <output file> <key>
```

Параметр key – целое число от 0 до 255, использующееся в алгоритме шифрования и дешифрования.

В качестве алгоритма шифрования должен использоваться следующий алгоритм, осуществляющий последовательно следующие операции с каждым байтом входного файла и записывающий результат в выходной файл:

1. Операция побитового XOR (исключающее ИЛИ, в Си/Си++ осуществляется при помощи оператора ^, например: 3 ^ 2 дает в результате 1, а 1 ^ 2 дает в результате 3) между байтом из файла и параметром key, переданном с командной строки.

2. Перемешивание битов в байте, полученном в результате предыдущей операции по следующей схеме (используйте операции для манипулирования битами):



3. Запись в выходной файл.

Алгоритм дешифрования идентичен алгоритму шифрования с той лишь разницей, что сначала выполняется перемешивание битов в обратном направлении, а затем операция XOR с параметром key, использованном при шифровании. Примечательной особенностью операции XOR является следующее свойство:

(A XOR B) XOR B = A XOR (B XOR B) = A XOR 0 = A

Некорректные входные данные (например, передача в качестве ключа строки, не являющейся десятичным числом, или числа, выходящего за пределы 0-255) должно выводиться соответствующе сообщение об ошибке. То же самое касается обработки ошибок открытия входного и выходного файлов.

Размер входного/выходного файлов ограничен 2 Гб.

В комплекте с программой должны обязательно поставляться файлы, позволяющие проверить ее работу в автоматическом режиме.

Задание 3

Выполните задание одного из предложенных варинатов.

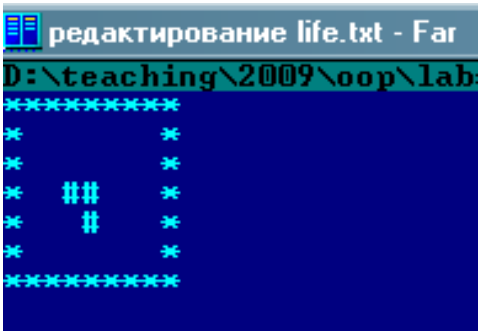
Вариант №1 – 150 баллов

Разработайте программу **live.exe**, моделирующую одну итерацию (расчет следующего поколения) клеточного автомата игры Жизнь.

Место действия этой игры — «вселенная» — это размеченная на клетки поверхность, безграничная, ограниченная, или замкнутая. **В случае данной лабораторной работы поверхность является ограниченной.** Каждая клетка на этой поверхности может находиться в двух состояниях: быть живой или быть мёртвой. Клетка имеет восемь соседей. Распределение живых клеток в начале игры называется первым поколением. Каждое следующее поколение рассчитывается на основе предыдущего по таким правилам:

- пустая (мёртвая) клетка рядом с тремя живыми клетками-соседями оживает;
- если у живой клетки есть две или три живые соседки, то эта клетка продолжает жить; в противном случае (если соседок меньше двух или больше трёх) клетка умирает (от «одиночества» или от «перенаселённости»).

Состояние первого поколения задается при помощи содержимого входного текстового файла следующего вида:



Символом * задаются границы поля (максимум, 256 * 256), а символом # задается положение живых клеток колонии.

Формат командной строки:

```
Life.exe <input file> [<output file>]
```

Если параметр <ouput file> не указан, то вывод следующего состояния колонии производится в стандартный поток вывода.

Внимание – на решение о том, выживет клетка, погибнет или родится, не должно влиять изменения состояния соседних клеток обработанных перед данной клеткой. Для этого можно использовать два двухмерных массива. Один массив с исходным состоянием колонии, заполняемый из входного файла, и массив со следующим состоянием колонии, рассчитывающийся по правилам игры на основе исходного массива и записывающийся в выходной файл (или выводящийся в output).

В комплекте с программой должны обязательно поставляться файлы, позволяющие проверить ее работу в автоматическом режиме.

Вариант №2 – 180 баллов

Разработайте программу **labyrinth.exe**, выполняющую поиск одного из возможных **кратчайших** путей между двумя точками в лабиринте, заданном в текстовом файле. Начальные и конечные точки задаются при помощи символов A и B. Найденный путь изображается при помощи символа «точка».

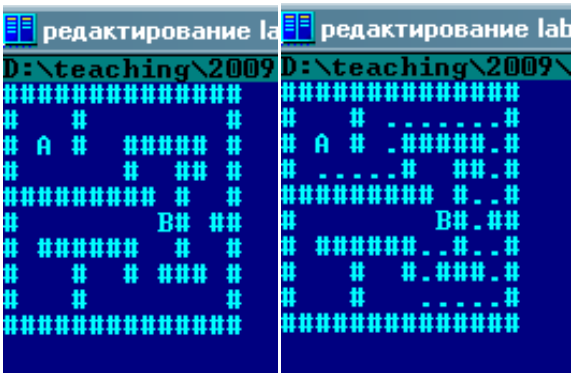
Лабиринт имеет максимальные размеры до 100*100 клеток и не обязательно должен иметь прямоугольную форму – задача приложения самостоятельно определить границы лабиринта, просканировав входной файл.

Во входном файле могут быть не заданы начальная и конечная точки, либо заданы больше чем 1 раз. При обнаружении такой ситуации приложение должно сообщить пользователю об ошибке.

При отсутствии пути между точками A и B необходимо вывести лабиринт без указания пути.

Формат командой строки:

```
labyrinth.exe <input file> <output file>
```

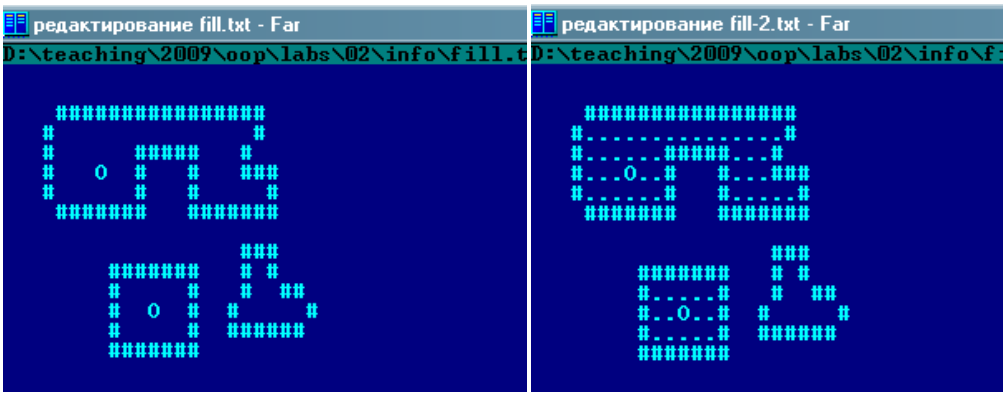


Для поиска кратчайшего пути можно использовать, например, Волновой Алгоритм.

В комплекте с программой должны обязательно поставляться файлы, позволяющие проверить ее работу в автоматическом режиме.

Вариант №3 – 180 баллов

Разработать приложение **fill.exe**, выполняющее заливку контуров, заданных в текстовом файле начиная с указанных начальных точек.



Максимальный размер изображения, заданного в текстовом файле – 100*100 точек.

Символ 'O' обозначает точки, начиная с которых должна выполняться заливка контуров.

Формат командной строки:

```
fill.exe <input file> <output file>
```

В комплекте с программой должны обязательно поставляться файлы, позволяющие проверить ее работу в автоматическом режиме.