Разработать ООП для формирования дампа любого заданного файла. Имена исходного файла и результирующего файла дампа должны передаваться программе через аргументы командной строки ее вызова. Исходный файл может иметь произвольный текстовый или бинарный формат. Результирующий файл дампа должен иметь текстовую структуру, идентичную стандартному выводу команды od из OS UNIX с ключом – x, где 2-х байтовые слова исходного файла кодируются короткими целыми числами без знака, которые записаны в системе счисления по основанию 16. При разработке программы нужно использовать методы бесформатного ввода и форматного вывода файловых потоков.

Файл дампа 16 имеет текстовый формат символьных строк фиксированной длины. Каждая строка кодирует 8 двухбайтовых слов целыми числами, которые разделены пробелом. Каждое число записано в системе счисления по основанию 16 и дополнено лидирующими нулями до 4 цифр. В начале каждой строки дампа указано смещение ее первого слова от начала исходного файла. Смещение измеряется в байтах и записано в системе счисления по основанию 8. Запись смещения дополнена лидирующими нулями до 7 цифр. Последняя строка дампа содержит запись размера исходного файла в формате, аналогичном смещению.

```
001:  #include <iostream>
002:  #include <fstream>
003:  #include <ios>
004:  #include <iomanip>
005:  using namespace std;
006:
007:  int main(int argc, char* argv[])
008:  {
009:    unsigned short i=0;
010:    unsigned offset = 0;
011:    if(argc < 2)
012:    {
013:      cerr << "Usage: asod source-file target-file" << endl;
014:      return(0);
015:    }
016:    ifstream input(argv[1]);
017:    if(input == 0)
018:    {
019:      cerr << argv[1] << ": Can't open to read" << endl;
020:      return(-1);
021:    }
022:    ofstream output;
023:    output.open(argv[2]);
024:    if(output == 0)
025:    {
026:      if(!argv[2])
027:      argv[2] = "target";
028:      cerr << argv[2] << ": Can't open to write" << endl;
029:      return(-2);
030:    }
031:    output.setf(ios::right, ios::adjustfield);
032:    output.fill('0');
033:    output << oct << setw(7) << offset;
034:    while(input.read((char *)&i, 2))
035:    {
036:      offset += 2;
037:      output << " " << hex << setw(4) << i;
038:      i = 0;
039:      if((offset % 16) == 0)
040:        output << endl << oct << setw(7) << offset;
041:    }
042:    if(input.gcount() == 1)
043:    {
044:      output << " " << hex << setw(4) << i;
045:      offset++;
046:    }
047:    output << endl;
048:    if(offset % 16)
049:    output << oct << setw(7) << offset << endl;
050:    input.close();
051:    output.close();
052:    return(0);
053:  }
```

Разработать ООП для восстановления информации по заданному дампу. Файл дампа должен иметь текстовую структуру , идентичную стандартному выводу команды od – x из OS UNIX. Для любого текстового или бинарного файла. Имена файлов дампа и результаы восстановления данных из него должны передаваться программе аргументами командной строки ее вызова7 программе должны применять методы форматного и бесформатного ввода-вывода файловых потоков в сочетании с потоковой обработкой строк символов.

```
001:  #include <iostream>
002:  #include <fstream>
003:  #include <ios>
004:  #include <iomanip>
005:  #include <sstream>
006:  #include <string.h>
007:  using namespace std;
008:
009:  int main (int argc, char* argv[])
010:  {
011:    unsigned short i=0;
012:    unsigned offset=0;
013:    static unsigned char word[2];
014:    if (argc<2)
015:    {
016:      cerr<<"Usage: dosa dump-file source-file"<<endl;
017:      return (0);
018:    }
019:    ifstream input(argv[1]);
020:    if (input==0)
021:    {
022:      cerr<<argv[1]<<" : Can't open to read"<<endl;
023:      return (-1);
024:    }
025:    ofstream output;
026:    output.open(argv[2]);
027:    if(output==0)
028:    {
029:      if(!argv[2])
030:      argv[2]="target";
031:      cerr<< argv[2]<<" : Can't open to read"<<endl;
032:      return(-2);
033:    }
034:    while(input>>oct>>offset)
035:    {
036:      input.setf(ios::hex, ios::basefield);
037:      while (input.get() != '\n')
038:      {
039:        input>>i;
040:        memcpy(word, &i, 2);
041:        output<<word;
042:      }
043:    }
044:    static char row[(48)]
045:    while (input.getline(row, 48))
046:    {
047:      string  s(row);
048:      istringstream sinput(s);
049:      sinput>>oct>>offset;
050:      sinput.setf(ios::hex, ios::basefield);
051:      while (sinput>>ws>>i)
052:      {
053:        memcpy (word, &i, 2);
054:        output<<word;
055:      }
056:    }
057:    input.close();
058:    output.close();
059:    return(0);
060:  }
```