ЛАБОРАТОРНАЯ РАБОТА № 17
ПОРАЗРЯДНЫЕ ОПЕРАЦИИ

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Цель работы – научиться работать с байтами и отдельными битами чисел, применять поразрядные логические операции и операции сдвига.

Поразрядные (или побитовые) операции

Применяются **только к целочисленным операндам** и работают с их двоичными представлениями. Таблица поразрядных операций приведена ниже:

Обозначение	Операция	Пример
~	поразрядное отрицание	~ X
&	поразрядное И	X & Y
|	поразрядное ИЛИ	X | Y
^	поразрядное исключающее ИЛИ	X ^ Y
<<	поразрядный сдвиг влево	X << 2
>>	поразрядный сдвиг вправо	Y >> 3

При выполнении логических поразрядных операций операнды сопоставляются **побитно** (первый бит первого операнда с первым битом второго, второй бит первого операнда со вторым битом второго и т.д.).

Поразрядные операции &, | и ^ работают в соответствии со следующей таблицей, где e1 и e2 — **сравниваемые биты** операндов:

e1	e2	e1 & e2	e1 | e2	e1 ^ e2
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	1	0

Поразрядное отрицание – унарная операция: инвертирует каждый разряд в двоичном представлении целочисленного операнда, например:

```
unsigned char a = 0x5D;  // 01011101 – в двоичной системе
unsigned char b = ~a;    // 10100010 = 0xA2
```

Не следует путать с логическим отрицанием (!), при котором результат будет равным 0 (т.е. false), если операнд не равен 0 (т.е. true) и, наоборот, равен 1 (true), если операнд равен 0 (false).

Поразрядная операция И – бинарная операция: бит результата равен 1 только тогда, когда соответствующие биты обоих операндов равны 1, например:

```
unsigned char a = 0x5D;  // 01011101 – в двоичной системе
unsigned char b = 0x36;  // 00110110 – в двоичной системе
unsigned char c = a & b; // 00010100 = 0x14;
```

Не следует путать с логическим И (&&), при котором результат будет равным 1 (т.е. true), только тогда, когда оба операнда не равны 0 (т.е. true).

Поразрядная операция ИЛИ – бинарная операция: бит результата равен 1 тогда, когда соответствующий бит хотя бы одного из операндов равен 1, например:

```
unsigned char a = 0x5D;  // 01011101 – в двоичной системе
unsigned char b = 0x36;  // 00110110 – в двоичной системе
unsigned char c = a | b; // 01111111 = 0x7F;
```

Не следует путать с логическим ИЛИ (||), при котором результат будет равным 1 (т.е. true), тогда, когда хотя бы один из операндов не равен 0 (т.е. true).

Поразрядная операция исключающее ИЛИ – бинарная операция: бит результата равен 1 только тогда, когда соответствующий бит **только** одного из операндов равен 1, например:

```
unsigned char a = 0x5D;  // 01011101 – в двоичной системе
unsigned char b = 0x36;  // 00110110 – в двоичной системе
unsigned char c = a ^ b; // 01101011 = 0x6B;
```

Операции сдвига – бинарные операции, применяются только к целочисленным операндам. Они предназначены для **сдвига целого числа вправо** или **влево** на заданное число двоичных разрядов. Первый операнд – сдвигаемое число, второй – количество разрядов, на которое это число нужно сдвинуть. При сдвиге влево (**<<**) освободившиеся разряды обнуляются. При сдвиге вправо (**>>**) освободившиеся биты заполняются нулями, если первый операнд беззнакового типа, и знаковым разрядом в противном случае. Например:

```
                          // для беззнаковых переменных
unsigned char a = 0x9D; // a = 10011101 – в двоичн. сист. счисл.
unsigned char b;
b = a << 1;  // значение a сдвигается влево на 1 разряд,
       // в результате b = 00111010 (значение a не изменилось!)
b = a << 2;  // b = 01110100 – сдвиг влево на 2 разряда
b = a >> 1;  // b = 01001110 – сдвиг вправо на 1 разряд
b = a >> 2;  // b = 00100111 – сдвиг вправо на 2 разряда

                  // для знаковых переменных
char c = 0x9D;    // c = 10011101 – в двоичн. сист. счисления
char d;
d = c >> 1;       // d = 11001110 – сдвиг вправо на 1 разряд
d = c >> 2;       // d = 11100111 – сдвиг вправо на 2 разряда
```

Операцию сдвига влево можно использовать вместо операции умножения на 2^k, а сдвига вправо – вместо деления на 2^k, где k – количество разрядов, на которое проводится сдвиг. Например:

```
short a = 12, b; // 00000000 00001100 – в двоичной системе
short b;
b = a << 2;      // 00000000 00110000 = 48   (12 * 2^2)
b = a >> 2;      // 00000000 00000011 =  3   (12 / 2^2)
```

В поразрядных логических операциях и операциях сдвига **операнды не изменяются**, т.е. после выполнения выражений

```
b = ~a; b = a << 2; b = a >> 2;
```

и т.п. переменная `a` **не изменяется**. Чтобы изменить саму переменную `a` (например, сделать сдвиг влево самой переменной `a`) необходимо записать:

```
a = a << 2
```

или в более коротком виде

```
a <<= 2.
```

Аналогично можно использовать и операции `>>=, &=, |=, ^=`.

Особенности использования поразрядных операций.

Поразрядные операции применяются при решении различных задач программирования, например работа с так называемыми битовыми флагами, шифрование информации, сжатие информации и т.д. Для этого нужно уметь выделять

определенный бит (или несколько битов) в двоичном представлении числа, устанавливать бит в 1 или сбрасывать в 0.

Выбор отдельных битов из целого числа осуществляется с помощью так называемых **битовых масок**. Битовая маска – целое число, которое используется для маскирования ненужных битов и выделения нужных, например, чтобы узнать значение пятого бита (считая слева) числа `10111011`, нужно использовать маску `00001000` и применить операцию **поразрядного И**. В результате получится:

```
10111011 & 00001000 = 00001000; // 5-ый бит равен 1
```

Чтобы установить заданный бит в `1` используют операцию **поразрядного ИЛИ** и соответствующую маску, например, чтобы в числе `10111011` установить второй слева бит в `1` нужно взять маску `01000000`:

```
10111011 | 01000000 = 11111011;
```

Чтобы сбросить заданный бит в `0` используют операцию **поразрядного И** и соответствующую маску, например, чтобы в числе `11101011` сбросить второй слева бит в `0` нужно взять маску `10111111`:

```
11101011 & 10111111 = 10101011;
```

Нетрудно заметить, что эта маска является **отрицанием** маски `01000000`:

```
~01000000 = 10111111;
```

На языке С это будет выглядеть так:

```
unsigned short a = 0x00BB; // 00000000 10111011 – в двоичн. системе
unsigned short mask = 0x0008; // 00000000 00001000 – в двоичн. сист.
unsigned short b, c;
b = a & mask; // выделение бита, получаем 00000000 00001000
mask = 0x0040; // новая маска - 00000000 01000000
b = a | mask;  // установили 10-й бит слева в 1,
               // получаем 00000000 11111011 = 0x00FB
c = b & ~mask; //  сбросили 10-й бит слева в 0,
               // получаем 00000000 10111011 = 0x00BB
```

Можно выделять, устанавливать или сбрасывать несколько бит сразу, для этого задаем соответствующую маску:

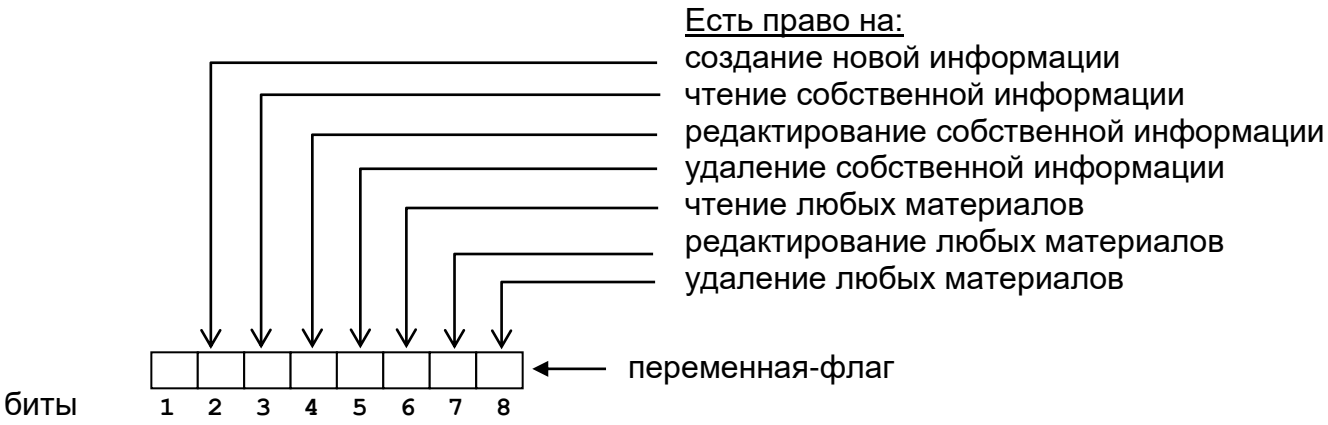
```
mask = 0x3C00; // новая маска - 00111100 00000000
 b = a | mask; // установили биты в 1,
               // получаем 00111100 10111011 = 0x2CBB
mask = 0x1800; // новая маска - 00011000 00000000
c = b & ~mask; // сбросили бит в 0,
               // получаем 00100100 10111011 = 0x18BB
```

Маски широко используются в программировании, например, в сетевых интернет-технологиях операция **поразрядного И** между значением IP-адреса и значением маски подсети используется для определения принадлежности данного адреса к подсети.

Пример 1. *Задание прав доступа с использованием битовой маски.*

Необходимо написать функции, которые позволяли бы задавать и контролировать различные права доступа пользователей к информации: создание новой информации, чтение собственной информации, редактирование собственной информации, удаление собственной информации, чтение любых материалов, редактирование любых материалов и удаление любых материалов.

Для решения данной задачи лучше всего подходит система битовых флагов. Для задания и хранения информации о правах пользователя будем использовать переменную-флаг, в которой каждый бит отвечает за свои права:



т.е. чтобы разрешить создание новой информации, чтение и редактирование собственной информации, нужно установить в 1 биты 2, 3 и 4 (получаем **01110000** или **0x70** – в 16-ричной системе). Длина слова, достаточная для хранения всех флагов, – один байт.

Объявим константы-флаги и создадим функции для задания и изменения флагов пользователя.

```
 typedef unsigned char BYTE;
     // Задаем константы-флаги – есть право на:
const BYTE F_NEW = 0x40;     // создание новой информации
const BYTE F_READOWN = 0x20; // чтение собственной информации
const BYTE F_EDITOWN = 0x10; // редактирование собственной инф-и
const BYTE F_DELOWN = 0x08;  // удаление собственной информации
const BYTE F_READALL = 0x04; // чтение любых материалов
const BYTE F_EDITALL = 0x02; // редактирование любых материалов
const BYTE F_DELALL = 0x01;  // удаление любых материалов

BYTE SetAccessRights()  // Задание прав доступа пользователя
{
  unsigned char Flag = 0;
  int f;

  printf("Разрешить создание новой инф-и (1 – да, 0 – нет)? ");
  scanf("%d", &f);
  if (f)
    Flag = F_NEW; // Задаем флаг
  printf("Разрешить обработку собственной инф-и (1–да, 0–нет)? ");
  scanf("%d", &f);
  if (f)
    Flag |= F_READOWN | F_EDITOWN | F_DELOWN; // Добавляем флаги
  printf("Разрешить обработку любых материалов (1–да, 0–нет)? ");
  scanf("%d", &f);
  if (f)
    Flag |= F_READALL | F_EDITALL | F_DELALL; // Добавляем флаги
  return Flag;
}

BYTE ChangeAccessRights(unsigned char Flag) // Изменение прав
{       // доступа пользователя
  int f;

  printf("Разрешить создание новой инф-и (1 – да, 0 – нет)? ");
  scanf("%d", &f);
```

```
  if (f)
    Flag |= F_NEW;    // Устанавливаем флаг
  else
     Flag &= ~F_NEW;  // Сбрасываем флаг
  printf("Разрешить обработку собственной инф-и (1-да, 0-нет)? ");
  scanf("%d", &f);
  if (f)
    Flag |= F_READOWN | F_EDITOWN | F_DELOWN; // Устанавливаем флаг
  else
     Flag &= ~(F_READOWN | F_EDITOWN | F_DELOWN); // Сбрас. флаг
  printf("Разрешить обработку любых материалов (1-да, 0-нет)? ");
  scanf("%d", &f);
  if (f)
    Flag |= F_READALL | F_EDITALL | F_DELALL; // Устанавливаем флаг
  else
     Flag &= ~(F_READALL | F_EDITALL | F_DELALL); // Сбрас. флаг
  return Flag;
}
```

Пример использования функций:

```
int main(int argc, char* argv[])
{
  unsigned char UserFlag;
  char UserName[20];

  printf("Введите имя пользователя: ");
  scanf("%s", UserName);
  UserFlag = SetAccessRights(); // Задание прав доступа польз.
// . . . . . .
  if (UserFlag & F_NEW)
    . . . // Можно сохранить информацию
// . . . . . .
  if (UserFlag & F_EDITOWN)
    . . . // Можно редактировать собственную информацию
// . . . . . .
  if (UserFlag & F_DELALL)
    . . . // Можно удалить любую информацию
// . . . . . .
  UserFlag=ChangeAccessRights(UserFlag); // Изменение прав доступа
// . . . . . .
  if (UserFlag & F_NEW)
    . . . // Можно сохранить информацию
// . . . . . .
  if (UserFlag & F_EDITOWN)
    . . . // Можно редактировать собственную информацию
// . . . . . .
  if (UserFlag & F_DELALL)
    . . . // Можно удалить любую информацию
// . . . . . .
  return 0;
}
```

Пример 2. *Простейшая функция шифрования информации.*

Простейший способ шифрования информации можно осуществить с помощью операции **поразрядного исключающего ИЛИ** и задаваемого пользователем кодового

слова. Будем осуществлять шифрование побайтно и длину кодового слова выберем равной 1 байту.

```
char Code;        // Кодовое слово
char Array[100]; // Шифруемый массив
```

Задаем кодовое слово (один символ):

```
scanf("%c", &Code);
```

Для шифровки выбираем очередной байт из массива или входного потока данных и преобразуем его с помощью операции поразрядного исключающего ИЛИ:

```
Array[i] ^= Code;
```

Например, пусть введено кодовое слово `0x5C` (`01011100` в двоичном представлении), а из массива взят байт `0x11`. После преобразования получаем:

```
0x11 ^ 0x5C = 0x4D;  // 00010001 ^ 01011100 = 01001101
```

Аналогично преобразуются другие байты. Для дешифровки используется та же процедура – операция **исключающее ИЛИ** с кодовым словом:

```
0x4D ^ 0x5C = 0x11;  // 01001101^ 01011100 = 00010001
```

Информация восстановлена. Если при дешифровке используется неверный код, информация будет восстановлена неправильно.

Создадим одну функцию для шифровки и дешифровки. В нее надо передать преобразуемый массив, его длину и кодовое слово:

```
void CodeDecode(char* Array, int Len, char Code)
{
   for(int i=0; i<Len; i++)
      Array[i] ^= Code;
}
```

Пример использования функций:

```
int main(int argc, char* argv[])
{
  int len;
  char* Arr;
  FILE *fp; // указатель на файл
  char Code;

  fp = fopen("Example.txt", "rb"); // Откр. двоичный файл на чтение
  if (!fp)
  {
    printf("File not found\n");
    return 0;
  }
  fseek(fp, 0L, SEEK_END);  // Переходим в конец файла
  len = ftell(fp); // Читаем позицию в файле - это будет длина файла
  fseek(fp, 0L, SEEK_SET);  // Переходим снова в начало файла
  Arr = new char[len];       // Выделяем память под массив
  fread (Arr, 1, len, fp);  // Читаем сразу весь массив
  fclose (fp);           // Закрываем файл
  printf("Input code: ");
  scanf("%c", &Code);    // Вводим код - один символ
  CodeDecode(Arr, len, Code); // Кодируем
  fp = fopen("Example.txt", "wb"); // откр. двоичный файл для записи
  fwrite (Arr, 1, len, fp);        // Записываем зашифрованный массив
  fclose (fp);                     // Закрываем файл
  delete[] Arr; // Освобождаем память
  return 0;
}
```

Для шифрования текста в файле запускаем программу и задаем кодовый символ, чтобы расшифровать файл, запускаем программу еще раз и задаем тот же кодовый символ.

Пример 4. *Вывод целого числа в двоичной системе счисления*

При использовании поразрядных операций удобно выводить числа в двоичной системе счисления, для этого необходимо проанализировать все разряды числа и если разряд окажется равен 0, то выводить на дисплей символ ‘0’, а если 1 – выводить символ ‘1’. Для анализа разрядов числа используем операцию **поразрядного И** и маску – специальную переменную, которая будет показывать, какой бит нужно анализировать.

Например, если маска равна `0x01`, то результат операции **И** с исследуемой переменной будет равен 1, если ее младший разряд равен 1, и 0, если он равен 0. Т.е. маска позволяет выделить нужный бит числа. Чтобы проанализировать разные биты числа, необходимо устанавливать соответствующую маску.

Напишем функцию, которая выводит значение переменной типа `short` в двоичной системе счисления. Размер переменной – два байта, следовательно, нужно проанализировать 16 бит. Назовем переменную `val` и введем переменную-маску `mask`. Начнем анализ со старшего бита, для этого присвоим маске значение `1000000000000000` (в двоичной системе или `0x8000` – в 16-ричной системе). Размер переменной-маски должен быть как минимум такой же, как исследуемой переменной, а для того, чтобы при сдвиге освобождающиеся биты заполнялись нулями, маска должна быть беззнакового типа (`unsigned short`).

Операция `val & mask` выделяет старший бит числа `val`. Если он равен 0, выводим на дисплей ‘0’, а если 1 – выводим ‘1’. Затем сдвигаем маску на 1 разряд вправо: `mask >>= 1`. Значение маски станет равным `0x4000`, следовательно, операция `val & mask` будет выделять второй справа бит числа `val`. Продолжаем процесс 16 раз (по числу бит числа `val`) последовательно выводя на дисплей символы ‘0’ или ‘1’.

```
void PrintBinary(short val)
{
  unsigned short mask = 0x8000; // 1000 0000 0000 0000

  for (int i = 0; i < 16; i++)
  {
    if (!(i % 4)) printf(" "); // Чтобы отделить по 4 бита
    if (val & mask)      // Выделяем очередной бит числа
      printf("%c", '1'); // Если он равен 1, выводим символ ‘1’
    else                 // иначе
      printf("%c", '0');  // выводим символ ‘0’
    mask >>= 1;     // сдвигаем маску на 1 бит вправо
  }
}
```

В качестве примера выведем число в десятичном, шестнадцатеричном и двоичном форматах. Чтобы отобразить число в десятичном и шестнадцатеричном формате проще всего воспользоваться функцией `printf` с соответствующими спецификаторами формата: `%d` – для десятичного, а `%x` – для шестнадцатеричного.

```
  printf("%d = %x = ", val, val);
```

где `val`– отображаемое число.

Чтобы отображать число в шестнадцатеричном формате так, как это принято в языке С (например, 15 отображается как 0x000F) надо в управляющую строку добавить `0x`:

```
  printf("%d = 0x%X = ", val, val);
```

Для выравнивания зададим в функции prinrf вывод 4 цифр для десятичного формата и 8 – для шестнадцатеричного.

```
printf("%4d = 0x%08X = ", val, val);
```

Для отображения в двоичном формате используем нашу функцию **PrintBinary**.

```
int main()
{
  short k;
  printf("k = ");                          // введем число 27
  scanf("%d", &k);
  printf("%4d = 0x%08X = ", k, k);
  PrintBinary(k);
  printf("\n");
  _getch();
  return 0;
}
```

```
k = 27
27 = 0x0000001B =  0000 0000 0001 1011
```

Для вывода числа в двоичном формате можно было бы использовать библиотечную функцию itoa(), но она не позволит вывести число так, как в нашей функции.

Пример 5. *Посчитать число битов равных 1 в двоичной записи числа типа short*

Чтобы посчитать число битов равных 1, нужно так же, как в предыдущем примере проанализировать по очереди все биты числа. Лучше всего написать отдельную функцию, в которую передать анализируемое число, а возвращать функция будет количество бит, равных 1. Для подсчета бит объявим переменную – счетчик, которую будем увеличивать, если очередной бит числа равен 1. Анализировать биты будем по тому же принципу, что и в предыдущем примере.

```
unsigned short HowOneBits(unsigned short a)
{
   unsigned short mask = 0x8000; // 1000 0000 0000 0000
   unsigned short k, counter = 0;  // Переменная – счетчик бит
   for (k = 0; k < 16; k++)
   {
     if (a & mask) // Если очередной бит равен 1
       counter++;  // увеличиваем счетчик
     mask >>= 1; // сдвигаем маску на 1 бит вправо
  }
  return counter;
}
```

Для иллюстрации работы функции напишем программу, в которой введем число и посчитаем число бит равных 1.

```
int main()
{
  int val, i;

  printf("val = ");
  scanf("%d", &val);
  i = HowOneBits(val);
  printf(" %d bit\n", i);
  _getch();
  return 0;
}
```

Например, если ввести число 125 ($125_{10} = 1111101_2$), то программа выведет надпись

```
" 6 bit"
```

Пример 6. *Посчитать общее число битов в двоичной записи числа типа short*

Число битов в числе зависит от самого числа, например:

$5_{10} = 101_2$ – 3 бита

$10_{10} = 1010_2$ – 4 бита

$48_{10} = 110000_2$ – 6 бит

$125_{10} = 1111101_2$ – 7 бит

Чтобы посчитать общее число битов числа, нужно так же, как в предыдущем примере анализировать по очереди все биты числа. Лучше всего написать отдельную функцию, в которую передать анализируемое число, а возвращать функция будет количество бит в числе. Анализировать биты будем в цикле по тому же принципу, что и в предыдущем примере, но сдвигать маску нужно до тех пор, пока не встретится бит, равный 1. Это и будет старший бит числа, а значение переменной цикла – номер этого бита.

```
unsigned short HowBits(unsigned short a)
{
  unsigned short mask = 0x8000; // 1000 0000 0000 0000
  unsigned short k;
  for (k = 0; k < 16; k++)
  {
    if ((a & mask) == 0) // Если очередной бит равен 0
      mask >>= 1;        // сдвигаем маску
    else      // если равен 1
      break;  // выходим из цикла
  }
  return 16 - k; // Эта разность и есть размер двоичной записи числа
}
```

Для иллюстрации работы функции напишем программу, в которой введем число и посчитаем число бит числа.

```
int main()
{
  int val, i;

  printf("val = ");
  scanf("%d", &val);
  i = HowBits(val);
  printf(" %d bit\n", i);
}
```

Например, если ввести число 125 ($125_{10} = 1111101_2$), то программа выведет надпись **" 7 bit"**

ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ

в начале программы ОБЯЗАТЕЛЬНО выводить:
ФИО, группа, номер лаб. работы, номер варианта.

Задания

Написать программу в соответствии с заданным вариантом. Программа должна быть написана с использованием функций. Вариант задания выдает преподаватель.

Для получения максимального балла нужно выполнить следующие требования:

– *Размер массивов должен задаваться с клавиатуры и память выделяться динамически. Программа должна повторяться ее до тех пор, пока не будет нажата клавиша ESC.*

– *Содержимое массивов должно задаваться с клавиатуры или случайными числами (по выбору пользователя)*

Для минимального балла достаточно сделать статические массивы и заполнить их случайными числами.

Варианты заданий

1. Разделить массив положительных целых двухбайтовых чисел на два других массива по числу элементов, содержащих четное число битов равных 1 и нечетное. Вывести массивы в десятичном, шестнадцатеричном и двоичном форматах. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размеры массивов и выделять под них память динамически. Заполнять исходный массив двумя способами: случайными числами и по порядку от x1 с шагом k. Находить максимальное и минимальное значения массивов. Выводить массивы и другую информацию на дисплей.

2. Удалить из массива положительных целых однобайтовых чисел элементы, содержащие заданные пользователем биты (например, если пользователь ввел 00010010, то нужно удалить элементы массива, которые содержат эти биты), сдвинув все оставшиеся элементы к началу массива (можно просто скопировать в другой массив). Вывести исходный и получившийся массивы в десятичном, шестнадцатеричном и двоичном форматах. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массива и выделять под него память динамически. Заполнять исходный массив двумя способами: случайными числами и по порядку от x1 с шагом k. Выводить массивы и другую информацию на дисплей.

3. В массиве положительных целых двухбайтовых чисел подсчитать для каждого элемента количество битов этого числа, которые равны 1. Удалить из массива элементы, у которых это посчитанное количество битов кратно 3, сдвинув все оставшиеся элементы к началу массива (можно просто скопировать в другой массив). Вывести массив в десятичном, шестнадцатеричном и двоичном форматах. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массива и выделять под него память динамически. Заполнять исходный массив двумя способами: случайными числами и по порядку от x1 с шагом k. Выводить массивы и другую информацию на дисплей.

4. В массиве положительных целых двухбайтовых чисел подсчитать для каждого элемента количество битов этого числа, которые равны 0. Найти минимальный элемент среди тех чисел, у которых это посчитанное количество битов кратно 3, и вывести его на дисплей в десятичном, шестнадцатеричном и двоичном форматах. Вывести массив в десятичном, шестнадцатеричном и двоичном форматах. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо

задавать размер массива и выделять под него память динамически. Заполнять исходный массив двумя способами: случайными числами и по порядку от x1 с шагом k. Выводить массивы и другую информацию на дисплей.

5. В массиве положительных целых двухбайтовых чисел подсчитать среднее арифметическое тех элементов, что содержат четное число битов не равных 1. Вывести массив в десятичном, шестнадцатеричном и двоичном форматах. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массива и выделять под него память динамически. Заполнять исходный массив двумя способами: случайными числами и по порядку от x1 с шагом k. Выводить массивы и другую информацию на дисплей.

6. Разделить массив положительных целых двухбайтовых чисел на два других массива по числу элементов, у которых число битов равных 1 кратно 3 и некратно. Вывести массивы в десятичном, шестнадцатеричном и двоичном форматах. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размеры массивов и выделять под них память динамически. Заполнять исходный массив двумя способами: случайными числами и по порядку от x1 с шагом k. Выводить массивы и другую информацию на дисплей.

7. В массиве положительных целых двухбайтовых чисел подсчитать для каждого элемента общее число битов в двоичной записи этого числа. Удалить из массива элементы, у которых это число битов нечетное, сдвинув все оставшиеся элементы к началу массива (можно просто скопировать в другой массив). Вывести исходный и получившийся массивы в десятичном, шестнадцатеричном и двоичном форматах. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массива и выделять под него память динамически. Заполнять исходный массив двумя способами: случайными числами и по порядку от x1 с шагом k. Находить максимальное и минимальное значения массивов. Выводить массивы и другую информацию на дисплей.

8. В массиве положительных целых двухбайтовых чисел найти минимальный элемент среди тех, что содержат четное число битов равных 1, и вывести его на дисплей в десятичном, шестнадцатеричном и двоичном форматах. Сдвинуть все элементы массива на 3 бита вправо и снова найти минимальный элемент среди тех, что содержат четное число битов равных 1. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массива и выделять под него память динамически. Заполнять исходный массив двумя способами: случайными числами и по порядку от x1 с шагом k. Выводить массивы и другую информацию на дисплей.

9. В массиве положительных целых двухбайтовых чисел подсчитать для каждого элемента общее число битов в двоичной записи этого числа. Подсчитать количество и сумму тех элементов, у которых это число битов кратно трем. Вывести массив в десятичном, шестнадцатеричном и двоичном форматах. Сдвинуть все элементы массива на 2 бита вправо и снова подсчитать количество и сумму элементов среди тех, что содержат четное число битов не равных 1. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массива и выделять под него память динамически. Заполнять исходный массив двумя способами: случайными числами и по порядку от x1 с шагом k. Выводить массивы и другую информацию на дисплей.

10. В массиве положительных целых однобайтовых чисел подсчитать количество и среднее арифметическое тех элементов, которые содержат заданные пользователем биты (например, если пользователь ввел 00010010, то нужно подсчитать элементы массива, которые содержать эти биты). Вывести массив в десятичном, шестнадцатеричном и двоичном форматах. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массива и выделять под него память динамически. Заполнять исходный массив двумя способами: случайными числами и по порядку от x1 с шагом k. Выводить массивы и другую информацию на дисплей.

11. Сравнить побайтно два массива положительных целых однобайтовых чисел и удалить из них те элементы, которые содержат одинаковое число бит равных 1 (можно не удалять, а просто обнулить их). Вывести массивы в десятичном, шестнадцатеричном и двоичном форматах. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять память под них динамически. Заполнять исходные массивы двумя способами: случайными числами и по порядку от x1 с шагом k. Выводить массивы и другую информацию на дисплей.

12. Переписать элементы массива A (целых двухбайтовых чисел со знаком) в массив B, сдвигая числа влево на число разрядов, заданное пользователем, и в массив C, сдвигая числа вправо на это же число разрядов. Вывести массивы в десятичном, шестнадцатеричном и двоичном форматах. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массивов и выделять память под них динамически. Заполнять исходные массивы двумя способами: случайными числами и по порядку от x1 с шагом k. Выводить массивы и другую информацию на дисплей.

13. В массиве положительных целых двухбайтовых чисел подсчитать для каждого элемента общее число битов в двоичной записи этого числа. Удалить из массива элементы, у которых это посчитанное число битов кратно 3, сдвинув все оставшиеся элементы к началу массива (можно просто скопировать в другой массив). Вывести массив в десятичном, шестнадцатеричном и двоичном форматах. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массива и выделять под него память динамически. Заполнять исходный массив двумя способами: случайными числами и по порядку от x1 с шагом k. Выводить массивы и другую информацию на дисплей.

14. В массиве положительных целых двухбайтовых чисел подсчитать для каждого элемента общее число битов в двоичной записи этого числа. Найти минимальный элемент среди тех чисел, у которых это посчитанное число битов кратно 3, и вывести его на дисплей в десятичном, шестнадцатеричном и двоичном форматах. Вывести массив в десятичном, шестнадцатеричном и двоичном форматах. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массива и выделять под него память динамически. Заполнять исходный массив двумя способами: случайными числами и по порядку от x1 с шагом k. Выводить массивы и другую информацию на дисплей.

15. Удалить из массива положительных целых двухбайтовых чисел элементы, содержащие четное число битов равных 1, сдвинув все оставшиеся элементы к началу массива (можно просто скопировать в другой массив). Вывести исходный и получившийся массивы в десятичном, шестнадцатеричном и двоичном форматах.

Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массива и выделять под него память динамически. Заполнять исходный массив двумя способами: случайными числами и по порядку от x1 с шагом k. Находить максимальное и минимальное значения массивов. Выводить массивы и другую информацию на дисплей.

16. В массиве положительных целых двухбайтовых чисел подсчитать количество и сумму элементов среди тех, что содержат четное число битов не равных 1. Вывести массив в десятичном, шестнадцатеричном и двоичном форматах. Сдвинуть все элементы массива на 2 бита вправо и снова подсчитать количество и сумму элементов среди тех, что содержат четное число битов не равных 1. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массива и выделять под него память динамически. Заполнять исходный массив двумя способами: случайными числами и по порядку от x1 с шагом k. Выводить массивы и другую информацию на дисплей.

17. Переписать из массива положительных целых однобайтовых чисел элементы, содержащие заданные пользователем биты (например, если пользователь ввел 00010010, то переписать нужно те элементы массива, которые содержать эти биты), сдвинув все оставшиеся элементы к началу массива (можно просто скопировать в другой массив). Вывести исходный и получившийся массивы в десятичном, шестнадцатеричном и двоичном форматах. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массива и выделять под него память динамически. Заполнять исходный массив двумя способами: случайными числами и по порядку от x1 с шагом k. Выводить массивы и другую информацию на дисплей.

18. В массиве положительных целых двухбайтовых чисел подсчитать для каждого элемента общее число битов в двоичной записи этого числа. Сдвинуть на 3 бита влево те элементы массива, у которых это число битов нечетное. Вывести массив в десятичном, шестнадцатеричном и двоичном форматах. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массива и выделять под него память динамически. Заполнять исходный массив двумя способами: случайными числами и по порядку от x1 с шагом k. Выводить массивы и другую информацию на дисплей.

19. Удалить из массива целых двухбайтовых чисел элементы, содержащие заданные пользователем биты (например, если пользователь ввел 00001111 11110000, то нужно удалить элементы массива, которые содержать эти биты), сдвинув все оставшиеся элементы к началу массива (можно просто скопировать в другой массив). Вывести исходный и получившийся массивы в десятичном, шестнадцатеричном и двоичном форматах. Программа должна содержать меню и позволять проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массива и выделять под него память динамически. Заполнять исходный массив двумя способами: случайными числами и по порядку от x1 с шагом k. Выводить массивы и другую информацию на дисплей.

20. В массиве положительных целых однобайтовых чисел найти минимальный элемент среди тех, что содержат четное число битов равных 1, и вывести его на дисплей в десятичном, шестнадцатеричном и двоичном форматах. Вывести массив в десятичном, шестнадцатеричном и двоичном форматах. Сдвинуть все элементы массива на 2 бита влево и снова найти минимальный элемент среди тех, что содержат четное число битов равных 1. Программа должна содержать меню и позволять

проводить повторные вычисления. Все алгоритмы должны быть реализованы с помощью функций. В программе необходимо задавать размер массива и выделять под него память динамически. Заполнять исходный массив двумя способами: случайными числами и по порядку от x1 с шагом k. Выводить массивы и другую информацию на дисплей.

Дополнительные варианты заданий

1. Написать функцию вывода целого числа в 4-ичной системе счисления (отделяя по 2 бита). Продемонстрировать на примере массива целых чисел. Вывести число в десятичном, шестнадцатеричном, 4-ичном и двоичном форматах. Размер массива вводится пользователем во время выполнения программы. Массив заполняется случайными числами.

2. Написать функцию вывода целого числа в 8-ичной системе счисления (отделяя по 3 бита). Продемонстрировать на примере массива целых чисел. Вывести число в десятичном, шестнадцатеричном, 4-ичном и двоичном форматах. Размер массива вводится пользователем во время выполнения программы. Массив заполняется случайными числами.

3. Дан массив из элементов типа unsigned char. Вывести массив в порядке возрастания по количеству бит равных 1. Вывести массив в десятичном, шестнадцатеричном и двоичном форматах. Размер массива вводится пользователем во время выполнения программы. Массив заполняется случайными числами.

4. Дан массив из элементов типа unsigned char. Вывести массив в порядке возрастания по количеству бит равных 0. Вывести массив в десятичном, шестнадцатеричном и двоичном форматах. Размер массива вводится пользователем во время выполнения программы. Массив заполняется случайными числами.

5. Написать программу, содержащую функции шифрования информации с помощью операции исключающее ИЛИ. Кодовое слово состоит из 8 десятичных цифр. Программа должна содержать меню и позволять: вводить кодовое слово с клавиатуры, задавать имя шифруемого файла и файла, куда должна выводиться зашифрованная информация, сохранять зашифрованные данные в заданный файл, читать зашифрованные данные из файла и выводить в расшифрованном виде на дисплей.

6. Написать программу, содержащую функции шифрования информации с помощью операции исключающее ИЛИ. Кодовое слово может содержать любые символы (не более 10). Программа должна содержать меню и позволять: вводить кодовое слово с клавиатуры, задавать имя шифруемого файла и файла, куда должна выводиться зашифрованная информация, сохранять зашифрованные данные в заданный файл, читать зашифрованные данные из файла и выводить в расшифрованном виде на дисплей.

7. Написать функцию для ввода целых однобайтовых чисел в двоичном формате. Написать программу для демонстрации функций, при этом выводить введенные числа в десятичном, шестнадцатеричном и двоичном форматах.

8. Написать функцию для ввода целых двухбайтовых чисел в двоичном формате. Написать программу для демонстрации функций, при этом выводить введенные числа в десятичном, шестнадцатеричном и двоичном форматах.

9. Написать функцию для ввода целых четырехбайтовых чисел в двоичном формате. Написать программу для демонстрации функций, при этом выводить введенные числа в десятичном, шестнадцатеричном и двоичном форматах.

10. Написать функцию для ввода в двоичном формате чисел с плавающей точкой (можно использовать объединение – union). Написать программу для демонстрации функций, при этом выводить введенные числа в десятичном, шестнадцатеричном и двоичном форматах.

11. Написать функцию для вывода в двоичном формате чисел с плавающей точкой (можно использовать объединение – union). Написать программу для демонстрации функций, при этом выводить введенные числа в десятичном и двоичном форматах.