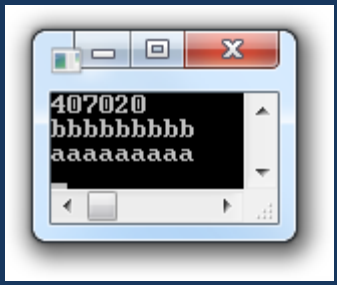
# Архитектурно-зависимое программирование

1. Работа с памятью на низком уровне (физическая память), см. «Указатели»
2. Учет особенностей компиляции, структуры кода и архитектуры: функции с переменных количеством параметров
3. Машинно-ориентированные (поразрядные) операции с машинными словами

# Особенности компиляции, структуры кода и архитектуры

Работа с адресами физической памяти: карта регистров, таблица векторов прерываний

```
    }
int globalData[1000];
#define REG0 *(char*)0x1000
#define REG *(char*)globalData
void F(){
    printf("%x\n",globalData);
    int val = REG;          // Регистры в АП = псевдо-переменные
    REG = 5;
    *(char*)0x407020 = 5;   // Записать 5 по адресу 0x407020
    //------------- Таблица векторов прерывания по известному адресу
    void (**tbl)();
    tbl = (void (**)())0x1000;  // Абстрактный ТД - преобразование ТД
    //------------ Проще - через typedef
    typedef void (**FPTRT)();   // ТД - указатель на МУ на функции
    typedef void (*FPTR)();     // ТД - указатель на функцию
    FPTRT tbl2 = (FPTRT)0x1000;
    FPTRT tbl3 = (FPTRT)new FPTR[10];
    tbl3[0]=F1;
    tbl3[1]=F2;
    (*tbl3[1])();
    (*tbl3[0])();
    }
```




```
void F1(){
    printf("aaaaaaaaa\n");
    }
void F2(){
    printf("bbbbbbbbb\n");
    }
```

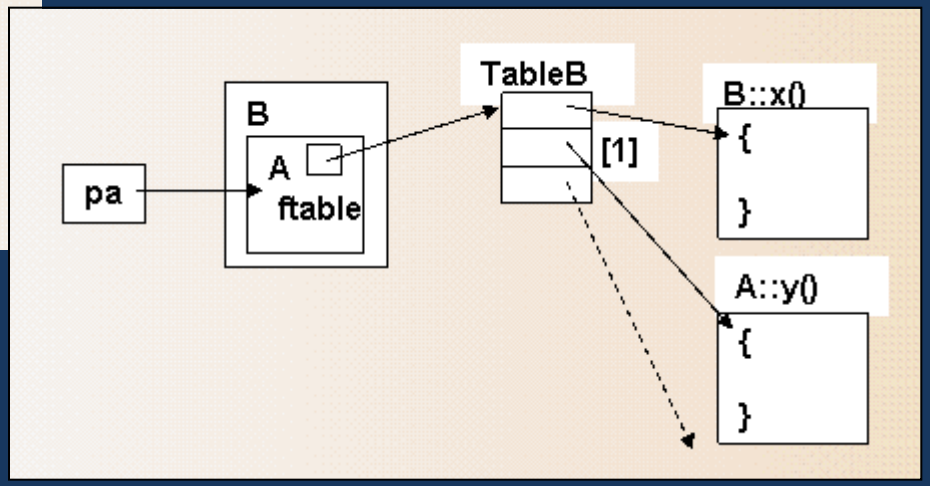
# Особенности компиляции, структуры кода и архитектуры

C++: виртуальная функция - в базовый класс пишется адрес таблицы функций (массива указателей на функции), которые использует ПРОИЗВОДНЫЙ КЛАСС. При вызове виртуальной функции через указатель на объект базового класса она будет вызвана в производном (полиморфизм) **иллюстрация**

```
// Выделены компоненты, создаваемые транслятором
class  A {
void   (**ftable)();        // Указатель на массив указателей
public:                     // на виртуальные функции (таблицу функций)
virtual void  x();
virtual void  y();
virtual void  z();
       A();
       ~A(); };
#define        vx      0        // Индексы в массиве
#define        vy      1        // указателей на виртуальные функции
#define        vz      2

// Таблица адресов функций класса A
void    (*TableA[])() = { A::x, A::y, A::z };

A::A()
{ ftable = TableA; ...}    // Назначение таблицы для класса A
```



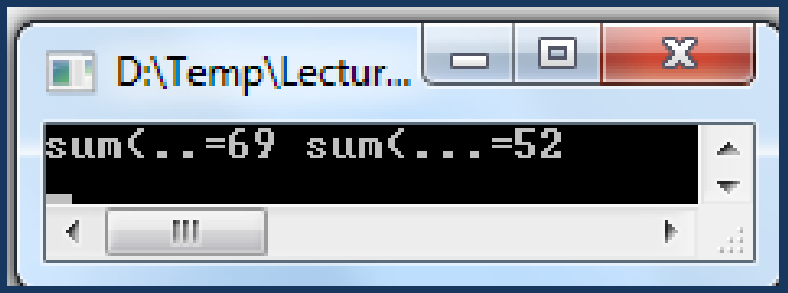
# Особенности компиляции, структуры кода и архитектуры

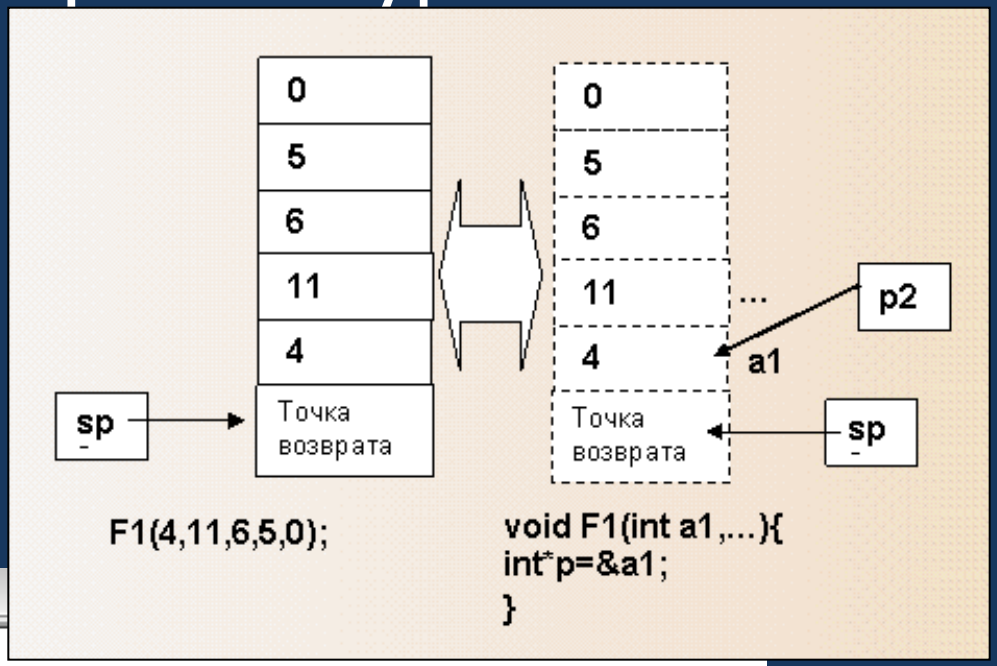
Работа со стеком: функции с переменным количеством параметров

1. Структура стека вызова: фактические (формальные) параметры, точка возврата, локальные переменные
2. Стек «вниз головой» - последний параметр в младших адресах, НО…
3. Компилятор записывает фактические параметры в стек В ОБРАТНОМ ПОРЯДКЕ, так что все располагается естественным образом в сегменте стека
4. Синтаксис void F(int a, int b, …) – переменный список фактических параметров НЕ МОЖЕТ БЫТЬ СИНТАКСИЧЕСКИ ПРОВЕРЕН КОМПИЛЯТОРОМ
5. F(int a, int b, …){ int *p; p=&b+1; } – по правилам адресной арифметики ссылается на область памяти (стек) «сразу после» b
6. Переменный список имеет заданный формат (саморазворачивающийся), в соответствии с которым функция извлекает данные
7. При нарушении формата в списке фактических параметров – НЕПРЕДСКАЗУЕМОЕ ПОВЕДЕНИЕ

# Особенности компиляции, структуры кода и архитектуры

Функции с переменным количеством параметров

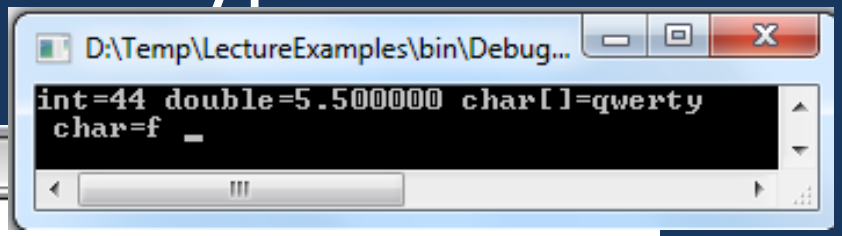



92-04.cpp

```
1   #include <stdio.h>
2   //-------------------------------------------------------92-04.cpp
3   //----- Сумма произвольного количества ненулевых параметров
4   int sum(int a,...){
5      int s,*p = &a;            // Указатель на область параметров назначается на
6      for (s=0; *p > 0; p++ )   // первый параметр из переменного списка
7          s += *p;              // Ограничитель - отрицательное
8      return(s); }              // значение
9   int main() {
10     printf("sum(..=%d sum(...=%d\n",sum(4,2,56,7,0),sum(6,46,-1 ,7,0));
11     getchar();
12     }
13
```

# Особенности компиляции, структуры кода и архитектуры




92-05.cpp

```
1  #include <stdio.h>
2  //-----------------------------------------------------92-05.cpp
3  //--- функция с параметром форматной строкой ( printf)
4   void my_printf(char *s,...)
5   { char *p = (char*)(&s+1);          // Указатель на начало списка параметров
6        while (*s != '\0') {           // Просмотр форматной строки
7        if (*s != '%') putchar(*s++); // Копирование форматной строки
8        else{
9              s++;                    // Спецификация параметра вида "%d"
10             switch(*s++){           // Извлечение параметра и переход к следующему
11       case 'c':   putchar(*p++); break;   // Извлечение символа
12       case 'd':   printf( "%d", *((int*)p));
13                   p+=sizeof(int);
14                   break;                  // Извлечение целого
15       case 'f':   printf( "%lf", *((double*)p));
16                   p+=sizeof(double);
17                   break;                  // Извлечение вещественного
18       case 's':   puts( *((char**)p));
19                   p+=sizeof(char*);              // Извлечение указателя
20                   break;                         // на строку
21                   }}}}
22 int main(){
23       my_printf("int=%d double=%f char[]=%s char=%c ",44,5.5,"qwerty",'f');
24       getchar();
25       }
```

# Поразрядные операции

Задачи:

- упаковка данных, архивирование. Естественно уплотнение данных производить с точностью не до байта, а до бита;
- алгоритмы сортировок и структуры данных, использующие элементы внутреннего представления хранимых данных: поразрядные сортировки, кодовые деревья;
- моделирование машинной арифметики. Если нас не устраивает точность представления данных, принятая в компьютере, мы сможем программно смоделировать данные более высокой точности. При этом можно пользоваться разными формами их представления, в том числе принятыми двоичными форматами данных;
- работа с растрами, битовыми картами (**bitmap**). Например, для контроля свободного пространства на диске файловая система часто использует карту памяти – массив байтов, каждый разряд которого определяется состояние отдельного блока: 0 – свободен, 1 – занят.

# Поразрядные операции

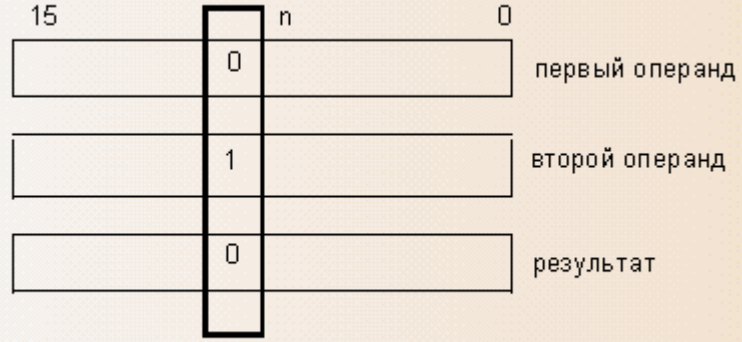
Матчасть:

- БТД – char,short,int,long = машинные слова размерности sizeof(…)
- Используются константы в 16СС – раскладка 1 цифра = 4 разряда
- Маска – константа в 16СС, используется для выделения разрадов, обрабатываемых в операции
- Логические и поразрядные операции: &&(&), ||(|), !(~) – первые работают с логической величиной 0/1, вторые - с парами разрядов
- Операции сдвига <<, >>
- Интерпретация операций с маской:
  - xxx & ccc – выделение разрядов по маске
  - xxx | ccc – установка разрядов по маске
  - xxx ^ ccc – инверсия разрядов по маске (исключающее И)
  - xxx & ~ccc – очистка разрядов по маске
- **Типичная ошибка** – приоритет операций (аналогично | & !, т.е. меньше, чем у сравнений)
- **Типичная ошибка** – расширение старшего (знакового) разряда для signed переменных и констант 0xFF -> 0xFFFFFFFF, гарантировано - 0x0FF
- **Типичная ошибка** – расширение старшего (знакового) разряда для signed переменных при сдвиге вправо (>>)
- **Типичная ошибка** – операции не меняют операндов, a = a >> 4; или a>>=4;

# Поразрядные операции

```
long     vv;                                  // Машинное слово двойной длины
for(int i=0; i<8*sizeof(long); i++)           // Количество разрядов в long
        { ... vv ... }                        // Цикл поразрядной обработки слова
```

```
0x1F -  0000 0000 0001 1111 (разряды 0..5 в 1)
0x3C0 - 0000 0011 1100 0000 (разряды 6..9 в 1)
0x1 -   0000 0000 0000 0001 (младший разряд)
```




```
x x x x x x x x  - операнд
0 0 1 1 1 0 0 0  - маска
0 0 x x x 0 0 0  - результат

b = a &  0x0861;        // Выделить разряды 0,5,6,11
b = a &  0x00F0;        // Выделить разряды с 4 по 7
                        // (разряды второй цифры справа)
```

```
x x x x  x x x x      - операнд
0 0 1 1  1 0 0 0      - маска
x x 1 1  1 x x x      - результат

a |= 0x0861;      // Установить в 1 разряды 0,5,6,11
a |= 0x00F0;      // Установить в 1 разряды с 4 по 7
                  // (разряды второй цифры справа)
```

```
a ^= 0x0861;      // Инвертировать разряды 0,5,6,11
a ^= 0x00F0;      // Инвертировать разряды с 4 по 7
                  // (разряды второй цифры справа)
```

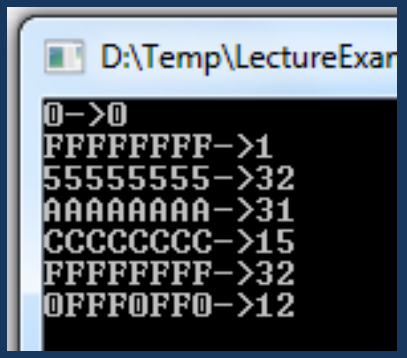
```
a &= ~0x0861;   // Очистить разряды 0,5,6,11, остальные сохранить
a &= ~0x00F0;   // Очистить разряды с 4 по 7, остальные сохранить
                // (разряды второй цифры справа)
```

# Поразрядные операции

```
//---------------------------------------------- 9
int F9(long n){ int     i,m,k;
    for (i=m=k=0; i < sizeof(long) * 8; i++, n >>= 1)
        if (n & 1) k++;
    else{
        if (k > m) m=k; k=0;
        }
    if (k > m) m=k; k=0;     // НЕТ В ТЕСТЕ
    return m; }
//                                                10
```

```
//---------------------------------------------- 8
// Операция "^" - ИСКЛЮЧАЮЩЕЕ ИЛИ
int F8(long n){
    int     i,m,k;
    for (i=m=k=0; i < sizeof(long) * 8; i++, n >>= 1)    // сдвиг МС после
        if ((n & 1) ^ m){            // Несовпадение мл.разряда и m
            k++;                     // Счетчик
            m =!m;                   // инверсия лог. 0-1-0-1
            }
return k; }
```

```
#include <stdio.h>
int main(){
    printf("0->%d\n",F8(0));
    printf("FFFFFFFF->%d\n",F8(0xFFFFFFFF));     // 1111111111111111
    printf("55555555->%d\n",F8(0x55555555));     // 0101010101010101
    printf("AAAAAAAA->%d\n",F8(0xAAAAAAAA));     // 1010101010101010
    printf("CCCCCCCC->%d\n",F8(0xCCCCCCCC));     // 1100110011001100
    printf("FFFFFFFF->%d\n",F9(0xFFFFFFFF));     //
    printf("0FFF0FF0->%d\n",F9(0x0FFF0FF0));     //
    }
```



# Поразрядные операции

Рекурсивное поразрядное разделение:

- разделение интервала массива на части по значению очередного разряда элементов
- на каждом шаге очередной разряд – от старшего к младшему

```
//-------------------------------------------------------91-04.cpp
//---------------- Поразрядная сортировка разделением
void      bitsort(int A[],int a,int b, unsigned m){
int i;
if (a   >= b) return;                          // Интервал сжался в точку
if (m   == 0) return;                          // Проверяемые разряды закончились
// Разделить массив на две части по значению разряда,
// установленного в маске m
int j,vv;                                     // Цикл разделения массива
for (i=a, j=b; i<=j; ){
        if ((A[i] & m) ==0)                    // Слева с разрядом =0
                { i++; continue; }             // пропустить
        if ((A[j] & m) !=0)                    // Справа с разрядом =1
                { j--; continue; }             // пропустить
        vv = A[i]; A[i]=A[j]; A[j]=vv;
        i++, j--;                             // Обмен и сдвиг границ
        }
bitsort(A,a,j,m >>1);                          // Рекурсивный вызов
bitsort(A,i,b,m >>1); }                        // для следующего разряда
```

```
void mainsort(int B[], int n){
int max,i; unsigned m;
for(max = 0, i=0; i< n; i++)
          if (B[i] > max) max = B[i];
for (m=1; m < max; m <<= 1);                  // Единичная маска, превышающая максимум
m >>=1;                                       // Старший разряд разделения
bitsort(B,0,n-1,m); }
```

# Поразрядные операции

Упаковка данных переменной размерности в битовом потоке

- Функции записи/чтения бита и слова в битовый поток (байтный массив)
- Номер бита передается по ссылке

```
#include <stdio.h>
//-------------------------------------------------------91-07.cpp
//-------- Извлечение и запись разряда (бита)
long getbit(char c[], int &n) { //c[] - массив байтов, n - номер разряда
    int nb = n/8;                    // номер байта
    int ni = n%8;                    // номер разряда в байте
    n++;
    return (c[nb]>>ni) & 1; }        //сдвинуть к младшему и выделить
void putbit(char c[], int &n, int v ){
    int nb = n/8;
    int ni = n%8;
    n++;
    c[nb] = c[nb] & ~(1<<ni) | ( (v&1) << ni);
    }
//------- Извлечение слова заданной размерности
unsigned long getword(char c[], int &n, int sz) {
    unsigned long v = 0;             // int sz - количество разрядов
    for(int i = 0; i<sz; i++)
        v |= getbit(c, n)<<i;
    return v;
    }
void putword(char c[], int &n, int sz, long v){
    while(sz--!=0){
        putbit(c, n, v&1);
        v>>=1;
        }
    }
```

# Поразрядные операции

```
//------- Извлечение слова заданной размерности
unsigned long getword(char c[], int &n, int sz) {
    unsigned long v = 0;              // int sz - ко
    for(int i = 0; i<sz; i++)
        v |= getbit(c, n)<<i;
    return v;
    }
void putword(char c[], int &n, int sz, long v){
    while(sz--!=0){
        putbit(c, n, v&1);
        v>>=1;
        }
    }
```

```
//---- Упаковка и распаковка переменных различной размерности
void unpack(char c[]){
int n=0; int vv;
   while(1){
     int mode=getword(c,n,2);          // Извлечение 2-разрядного кода
     switch(mode){                     // переключение по типу переменной
   case 0: return;
   case 1: vv=getword(c,n,8); break;   // 01 извлечь 8-разрядное (байт)
   case 2: vv=getword(c,n,16);break;   // 10 извлечь 16-разрядное
   case 3: vv=getword(c,n,32);break;   // 11 извлечь 32-разрядное
       }
     printf("%d ",vv);
     }
   printf("\n");
   }
```

# Поразрядные операции

```
int pack(char c[], int dd[], int nn){
    int n=0; int vv;
    for(int i=0;i<nn;i++){
        vv = dd[i];
        if (vv < 256) {
            putword(c,n,2,1);    // запись 2-разрядного кода 01
            putword(c,n,8,vv);
            }                    // запись 8-разрядного кода числа
        else
        if (vv < 32768) {
            putword(c,n,2,2);    // запись 2-разрядного кода 10
            putword(c,n,16,vv);
             }                   // запись 16-разрядного кода числа
        else {
            putword(c,n,2,3);    // запись 2-разрядного кода 11
            putword(c,n,32,vv); // запись 32-разрядного кода числа
            }
        }
      putword(c,n,2,0);          // Концевик для 0
      return n/8;
      }
```

# Поразрядные операции

Упаковка данных переменной размерности в битовом потоке (оснастка)

```
void putmem(char cc[], int nn){
    for(int i=0;i<nn;i++){
        if (i%8==0)
            printf("\n");
        printf("%2x ",cc[i] & 0x0FF);
        // Знаковое расширение при char->int
        }
    printf("\n");
    }
int main(){
    char cc[100];
    int vv1[]={5,6,7,8};
    int mm = pack(cc,vv1,4);
    printf("sz=%d\n",mm);
    putmem(cc,mm+1);
    unpack(cc);
    }
```

**01** 00000101 **01** 00000110 **01** 000001111 **01** 000010000 00

```
sz=5

15 64 d0 41  8 fc
5 6 7 8
```

0 1 2 3

0001_0101 0110_0100 1101_0000 0100_0001 0000_1000 1111_1100 4

# Поразрядные операции

Пример: упаковка групп медленно меняющихся данных (временной ряд) в битовую последовательность (Java):

- delta – кодирование, сохраняется приращение, а не абсолютное значение
- Формат – бит смены размерности v{1} =0 n-разрядов переменной с предыдущей размерностью, v{1} =0 sz = s{5}+1  sz  - 5 разрядов новой размерности
- Оснастка – функции чтения и записи групп разрядов из потока

```
//-------- Извлечение и запись разряда (бита)
private int getbit() {
    int nb = bitIdx / 8;        // номер байта
    int ni = bitIdx % 8;        // номер разряда в байте
    bitIdx++;
    return (in[nb] >> ni) & 1;  //сдвинуть к младшему и выделить
    }
private void putbit(int v) {
    int nb = bitIdxOut / 8;
    int ni = bitIdxOut % 8;
    bitIdxOut++;
    one = (byte)(one & ~(1 << ni) | ((v & 1) << ni));
    if (bitIdxOut%8==0){
        out.write(one);
        one=0;
        }
    }
//------- Извлечение
```

```
private void putword(int nBit, int v) {
    while (nBit-- != 0) {
        putbit( v: v & 1);
        v >>= 1;
        }
    }
```

```
//------- Извлечение слова заданной размерности
private int getword(int nBit) {
    int v = 0;
    int i=0;
    int lastBit=0;
    for (i = 0; i < nBit; i++){
        lastBit = getbit() << i;
        v |= lastBit;
        }
    while(i++!=32){
        lastBit <<=1;
        v |= lastBit;
        }
    return v;
    }
```

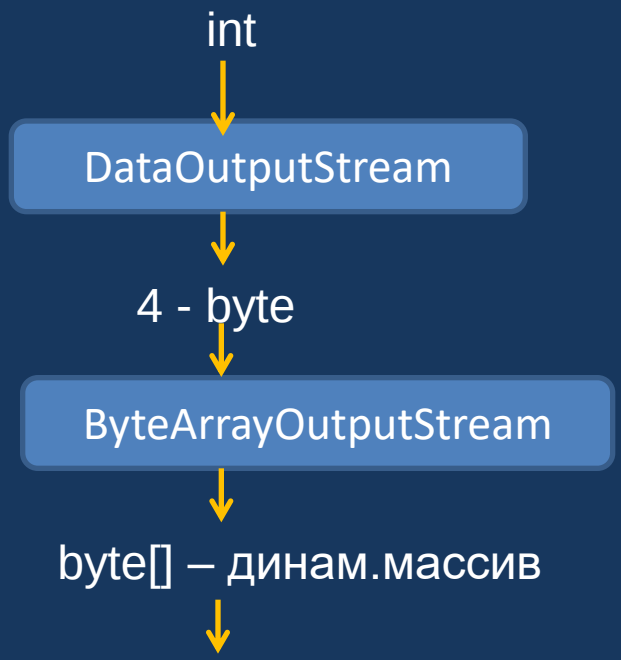
```
private int wordLen(int v){     // Разрядность слова
    if (v<0)                    // Абсолютная величина
        v = -v;
    int k=0;
    while(v!=0){
        k++;
        v >>=1;
        }
    return k;                   // Без знака
    }
```

# Поразрядные операции

```
public byte[] decode(byte []in0, int sz) throws IOException {
    in = in0;
    ByteArrayOutputStream bout = new ByteArrayOutputStream();
    DataOutputStream out = new DataOutputStream(bout);
    bitIdxOut=0;
    bitIdx=0;
    oldLnt=-1;
    for(int i=0;i<sz;i++){              // 1 бит-индикатор смены длины
        int v0 = getbit();              // 5 бит - новая длина
        if (debug)
            System.out.println("i="+i+" v0="+v0);
        if (v0!=0){                     // Само слово
            oldLnt = getword( nBit: 5)+1;
            if (debug)
                System.out.println("lnt="+oldLnt);
            }
        int v2 = getword(oldLnt);
        if (debug)
            System.out.println("word="+v2);
        int v3 = v2;
        if (deltaMode && i!=0){
            v3 = prevValue + v2;
            }
        prevValue = v3;
        out.writeInt(v3);
        }
    out.flush();
    byte bb[] = bout.toByteArray();
    bout.close();
    out.close();
    return bb;
    }
```

# Поразрядные операции

```
public byte[] decode(byte []in0, int sz) throws IOException {
    in = in0;
    ByteArrayOutputStream bout = new ByteArrayOutputStream();
    DataOutputStream out = new DataOutputStream(bout);
    bitIdxOut=0;
    bitIdx=0;
    oldLnt=-1;
    for(int i=0;i<sz;i++){              // 1 бит-индикатор смены длины
        int v0 = getbit();              // 5 бит - новая длина
        if (debug)
            System.out.println("i="+i+" v0="+v0);
        if (v0!=0){                     // Само слово
            oldLnt = getword( nBit: 5)+1;
            if (debug)
                System.out.println("lnt="+oldLnt);
            }
        int v2 = getword(oldLnt);
        if (debug)
            System.out.println("word="+v2);
        int v3 = v2;
        if (deltaMode && i!=0){
            v3 = prevValue + v2;
            }
        prevValue = v3;
        out.writeInt(v3);
        }
    out.flush();
    byte bb[] = bout.toByteArray();
    bout.close();
    out.close();
    return bb;
    }
```




- Буферизация
- Запись в поток (память) в переменном формате

```
20:03:19 20:02:51 <——109.174.113.128/api/ess/registers/list/read POST  time=1 мс объектов 0 (0)
20:03:19 20:02:55 Потоковые данные редкие =3440 сжатие=0,066(0,114) 12 мс
20:03:19 20:02:58 Потоковые данные обычные =34 сжатие=1,000(0,115) 8 мс
20:03:19 20:03:01 <——109.174.113.128/api/ess/register/value/read GET regnum=500 time=0 мс объект
20:03:19 20:03:01 Группы регистров: 492[1] 510[3] 520[1] 554[4] 560[4] 566[34] 620[6] 630[4] 652[1] 660[
20:03:19 20:03:01 <——109.174.113.128/api/ess/registers/list/read POST  time=2 мс объектов 0 (0)
20:03:19 20:03:09 Потоковые данные обычные =34 сжатие=1,000(0,116) 9 мс
```

# Поразрядные операции

Арифметика с целыми неограниченной точности (точные вычисления)

- Во внутренней форме – массив байтов = машинное слово
- Во внешней форме – строка символов-цифр  (10 СС)
- В смешанной форме – массив байтов, каждая тетрада = цифра (BDC-код, двоично-десятичное представление)

Алгоритмы:

Внешняя и смешанная:

- Сложение: пар цифр с переносом (в столбик)
- Вычитание: представление отрицательных чисел – инверсия цифр + 1

  -125 = -0125 = 9874 +1 = 9875

  +068 = 0068

  -125 +068 = 9875+0068 = 9943 = -0056+1 = -057

  -125 +236 = 9875+0236 = 1|0111 = +111
- Умножение: ∑∑ai*bj * 10^(i+j) – сумма произведений: перемножить пару цифр, получить 2 цифры произведения, дописать справа i+j нулей, добавить к сумме.
- Деление:…

# Поразрядные операции

```
//-------------------------------------------------------------------91-12.cpp
//------ Инкремент числа во внешней форме представления
void inc(char s[]){
for (int i=0; s[i]!=0; i++);                    // Поиск конца строки
for (int n=i-1; n>=0; n--){                      // Младшая цифра - в конце
     if (s[n]=='9' )                            // 9 превращается в 0
                    s[n]='0';
     else { s[n]++; return; }}                  // добавить 1 к цифре и выйти
for (s[i+1]=0; i>0; i--) s[i]='0' ;             // Записать в строку 1000...
s[0]='1';}
```

Во внешней форме

```
long ss=0x00017655;
char s[]={0x55,0x76,0x10,0x00};
char s[]="17665";
```

```
//-------------------------------------------------------------------91-13.cpp
//------ Сложение чисел во внешней форме представления
 void add(char out[],char c1[],char c2[]){
 int l1=strlen(c1)-1;                          // Определение разрядности суммы
 int l2=strlen(c2)-1;                          // и индексов младших цифр слагаемых
 int l=l1; if (l2>l1) l=l2;
 l++; out[l+1]=0;                                    // В сумме на 1 цифру больше
 int v,v1,v2,carry;                           // Цифры и перенос
     for (carry=0;l>=0;l--,l1--,l2--){         // Цикл от младших цифр к старшим
     if (l1<0) v1=0; else v1=c1[l1]-'0';
     if (l2<0) v2=0; else v2=c2[l2]-'0';
     v=v1+v2+carry;                                  // Сложение с учетом входного
           if (v>=10) {carry=1; v-=10;}      // и формированием выходного
     else carry=0;                           // переноса (во внутренней форме)
     out[l]=v+'0';                           // Запись цифры результата
     }}
```

# Поразрядные операции

Во внутренней форме

```
#include <stdio.h>
typedef unsigned char uchar;
 void add(uchar out[], uchar in1[], uchar in2[], int n){
    int i, carry;                    // Бит переноса
    unsigned w;                    // Рабочая переменная для
    for (i=0, carry=0; i<n; i++){
        out [i] = w = in1[i]+in2[i]+carry;
        carry = (w & 0x0100) >>8;
        }}
 void neg(uchar in[], int n){
     int i, carry;
    unsigned w;
    for (i=0; i<n; i++) in[i]=~in[i];
    for (i=0, carry=1; i<n; i++){
      in [i] = w = in[i]+carry;
      carry = (w & 0x0100) >>8;
      }}
```

```
  //------Сдвиг целых произвольной разря
 void lshift(uchar in[], int n){
    int carry;                        //
    int i,z;
    for (carry=0, i=0; i<n; i++){
        z=(in[i] & 0x80)>>7;          // Выделить старший разряд (перенос)
        in[i] <<= 1;                  // Сдвинуть влево и установить
        in[i] |=carry;                // старый перенос в младший разряд
        carry = z;                    // Запомнить новый перенос
        }}
 void rshift(uchar in[], int n){      // Сдвиг арифметический
    int carry=((in[n-1]&0x80)!=0);    // Разряд переноса = копия знакового
    int i,z;
    for (i=n-1; i>=0; i--){
        z = in[i] & 1;                // Выделить младший разряд (перенос)
        in[i] >>= 1;                  // Сдвинуть вправо и установить
        in[i] |= carry <<7;           // старый перенос в старший разряд
        carry = z;                    // Запомнить новый перенос
        }}
 //-------------------------------------------------------91-11.cpp
```

# Поразрядные операции

Во внутренней форме

```
//-----------------------------------------------------91-11.cpp
//------Умножение целых произвольной разрядности
 void mul(uchar out[], uchar aa[], uchar bb[], int n)
 {int i,s1=0,s2=0;                          // Отрицательные числа - прямой код
 if (aa[n-1] & 0x80) { neg(aa,n); s1=1; }
 if (bb[n-1] & 0x80) { neg(bb,n); s2=1; }
 for (i=0; i<n; i++) out[i]=0;
      for (i=0; i< n* 8; i++){              // Цикл по количеству разрядов
      if (bb[0] & 1 )                       // Разряд множителя равен 1
      add(out,out,aa,n);                    // Добавить множимое к произведению
      lshift(aa,n);                         // Множимое - влево
      rshift(bb,n);                         // Множитель - вправо
      }
 if (s1!=s2) neg(out,n);                    // Знаки не совпадают - доп. код
 }

void main(){
long a=125, b=300, c;
mul((uchar*)&c, (uchar*)&a, (uchar*)&b,sizeof(long));
printf("c=%ld\n",c);
long a1=-125, b1=300;
mul((uchar*)&c, (uchar*)&a1, (uchar*)&b1,sizeof(long));
printf("c=%ld\n",c);
}
```