ЛЕКЦИЯ 6.

Тема: Алгоритмы обработки чисел

1. Алгоритмы обмена

Поменять местами значения двух переменных *a* и *b*.

a)
```
int a,b,tmp;
   tmp=a; a=b; b=tmp;
```

b)
```
a=a+b; b=a-b; a=a-b;
```

c)
```
a=a^b; b=a^b; a=a^b;
```

2. Нахождение НОД двух натуральных чисел (a,b)

Алгоритм Евклида

b) Если *a* „ у *b* не равные одновременно нулю (по условию числа натуральные), то можно вычислить НОД следующим образом:

$a = bx_1 + r_1$

$b_1 = r_1x_2 + r_2$

$b_2 = r_2x_3 + r_3$

...

$b_{n-1} = r_{n-2}x_{n-1} + r_{n-1}$

$b_n = r_{n-1}x_n + 0$ => r_{n-1} = „ N„ O„ D(a,b)

Запишем этот алгоритм на псевдокоде:

Дана пара натуральных чисел (a, b).
Шаг 1. Поделить a на b с остатком r.
Шаг 2. Если r = 0, то НОД (a, b) = b.
Шаг 3. Если r <>0, то положить a=b, b=r и перейти к шагу 1.

Соответствующий фрагмент программы:

```
int a,b,r;
cout << "input 2 numbers\n";
cin >> a >> b;
while (b != 0)
   {
     r = a % b;
     a = b;
     b = r;
   }
cout << "NOD=" << a;
```

Можно реализовать и по-другому:

```
do
  if (a>b)
      a=a % b;
  else
      b=b % a;
while ((a !=0) && (b!=0));
```

3. Бинарный алгоритм Евклида для нахождения НОД целых чисел

Вычисления проводятся на основании следующих свойств НОД:

НОД(2m,2n) = 2 НОД(m ,n),
НОД(2m,2n+1) = НОД(m,2n+1),
НОД(-m,n) = НОД(m,n)
Имеем:
НОД(0, n) = n; НОД(m, 0) = m;
Если m,n чётные, тогда НОД(m, n) = 2*НОД(m/2, n/2)
Если m чётное, тогда НОД(m, n) = НОД(m/2, n)
Если n чётное, тогда НОД(m, n) = НОД(m, n/2).
Если m,n нечётные и m > n, тогда
НОД(m, n) = НОД(m - n, n)
Если m,n нечётные и m < n, тогда НОД(m, n) = НОД(n - m, m)
Если m = n, тогда НОД(m, n) = m;

```
#include <iostream>
using namespace std;
int main()
{
   setlocale(LC_ALL,".1251");
   int shift;
   int x,y,nod;
   cout << "введите 2 числа\n";
   cin >> x >> y;
   if (x == 0 || y == 0) //одно из чисел=0
     nod = x | y; // тогда НОД=ненулевому
   else
   {
     /* shift = наибольшая степень 2   такая, чтобы x и y
   разделились на 2   в этой степени */
     for (shift = 0; ((x | y) & 1) == 0;
              ++shift)
     {// цикл работает пока
      // оба числа четные
         x >>= 1; // деление на 2
         y >>= 1; // деление на 2
```

```
    }
    while ((x & 1) == 0)
          // пока  x не станет нечетным
          x >>= 1; // делим на 2
    do
    {
        while ((y & 1) == 0) //для y тоже
            y >>= 1; // делим на 2
        /* теперь x и y are оба нечетны,
         разность их четная,
         положим x = min(x,y),
         y = разности большего и меньшего, деленной на 2 */
        if (x < y)
        {
            y -= x;
        }
        else
        {
            unsigned int diff = x - y;
            x = y;
            y = diff;
        }
        y >>= 1;
    }
    while (y != 0);
        nod = x << shift;
  }
  cout << "NOD=" << nod;
return 0;
}
```

4. Переворачивание целого числа

```
    int a, b, ost, s=0;
    a=12345;
    b=a;
    while(b>0)
    {
        ost=b%10;
        b=b/10;
        s=s*10+ost;// s=54321;
    }
```

5. Найти количество повторений каждой цифры у заданного натурального числа (не используя массивов).

```
{
     int a, b, c;
     cout << ”input number”;
     cin >> a;
     b = a;
     for(int i = 0; i<10; ++i)
     {
          c = 0;
          a = b;
          while(a>0)
          {
               if(a%10 == i) c++;
               a /=10;
          }
          if(c!=0)
               cout << i << ”-” << c;
     }                                                    }
```

6. Нахождение всех делителей натурального числа n

```
//Все делители натурального числа n
#include <iostream>
using namespace std;
int main()
{
     setlocale(LC_ALL,".1251");
     long int n,d,k;
     cout << "введите число" << endl;
     cin >> n;
     cout << "1" << endl;
     for(d=2,k=n/2; d<=k; d++)
     {   if (n%d==0)
               cout << d << endl;
     }
     cout << n << endl;
     return 0;
}
```

7. Проверка числа на простоту

```
int a, i, flag; double r, n;
cin >> a;
if ((a==2)||(a==3)) flag=1;
else
{ i=2;
  flag=1;
  r=a; n=(floor(sqrt(r));
  while ((i<=n) && (flag))
   if (!(a%i)) flag=0;
   else ++i;
 if flag cout << "простое";
 else cout << "составное";
}
```

Функция floor(x) возвращает значение с плавающей точкой, представляющее наибольшее целое, которое меньше или равно x.

8. Нахождение 3-х значных чисел без повторяющихся цифр (без использования операций деления)

```
// 3-х значные числа без повторений цифр
int main()
{
int i,j,k;
for(i=1;i<=9;i++)
 for(j=0;j<=9;j++)
  for(k=0;k<=9;k++)
    if((i!=j)&&(i!=k)&&(j!=k))
     cout << i*100+j*10+k << " ";
return 0;
}
```

9. Вычисление квадратного корня числа по итерационной формуле с заданной точностью

$$x_n = \frac{x_{n-1}}{2} + \frac{a}{2 * x_{n-1}} \qquad x_0 = \frac{a}{2}$$

```
#include <cmath>
...
{   double a, e, x, xn;
    cout << "число и точность\n";
    cin >> a >> e;
    xn=a/2;   // Значение x0
    do
```

```
        {    x=xn;
             xn=x/2+a/(2*x);
        }
    while (abs(xn-x)>e);
    cout << x << endl;
    return 0;
}
```

10. Числа Мерсенна

В XVII веке французский математик М. Мерсенн определил последовательность чисел вида: $\mathbf{M_n = 2^n - 1}$, n=1,2,…

Эта последовательность получила наименование «**чисел Мерсенна**»:
1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, ...

Иногда числами Мерсенна называют числа M_p с простыми индексами p.

Эта последовательность: 3, 7, 31, 127, 2047, 8191, 131071, 524287, 8388607,...

До 1750 года было найдено всего 8 простых чисел Мерсенна: M2, M3, M5, M7, M13, M17, M19, M31.

То, что M31 - простое число, доказал в 1750 году Л. Эйлер.

В 1876 году французский математик Эдуард Люка установил, что число M127=170141183460469231731687303715884105727- простое.

В 1883 г. Сельский священник Пермской губернии И.М.**Первушин** без всяких вычислительных приборов доказал, что число M61=**2305843009213693951** является простым. Позднее было установлено, что числа M89 и M107 -простые.

Использование ЭВМ позволило в 1952-1964 годах доказать, что числа M521, M607, M1279, M2203, M2281, M3217, M4253, M4423, M2689, M9941, M11213- простые.

```
{//поиск чисел Мерсенна
 int m=10;
 for (int i=1;i<=m;i++)
 {
  int r=1,x=2;
  int n=i;
  while  (n != 0)
  {
    if (n % 2 == 1) r=r*x;
    x=x*x;
    n=n/2;
  }
  r=r-1;
  cout << r << endl;
 }
}
```

11. Совершенные числа

Натуральное число P называется совершенным, если оно равно сумме всех своих делителей кроме P.

Евклид доказал, что если p и 2p-1 - простые числа, то число $P=2^{p-1}(2^p-1)$ является совершенным.

Первые четыре совершенных числа приведены в *Арифметике* **Никомаха Геразского**.

Числа P2=6 и P3=28 были известны ещё пифагорейцам.

Числа P5=496 и P7=8128 нашел Евклид.

Пятое число **P13** =**33550336** обнаружил немецкий математик **Регионтан** (XV век).

В XVI веке немецкий ученый **Шейбель** нашел еще два числа:

P17 =8589869056 и P19= 137438691328.

P31=2305843008139952128.

В начале XX в. были найдены еще 3 совершенных числа (для p = 89, 107 и 127). В дальнейшем поиск затормозился вплоть до середины XX в., когда с появлением компьютеров стали возможными огромные вычисления.

На февраль 2008 г. известно 44 чётных совершенных числа.

Нечётных совершенных чисел до сих пор не обнаружено, однако не доказано и то, что их не существует.

Неизвестно также, бесконечно ли множество всех совершенных чисел.

До сих пор остаётся загадкой, как Мерсенн смог высказать правильное утверждение, что числа P17, P19, P31 являются совершенными. Позднее было обнаружено, что почти за сто лет до Мерсенна числа P17, P19 нашел итальянский математик Катальди - профессор университетов Флоренции и Болоньи.

Считалось, что божественное провидение предсказало своим избранникам правильные значения этих совершенных чисел.

Пифагорейцы считали первое совершенное число 6 символом души, второе совершенное число 28 соответствовало числу членов многих учёных обществ, в двенадцатом веке церковь учила: для спасения души достаточно изучать совершенные числа и тому, кто найдёт новое божественное совершенное число, уготовано вечное блаженство, то становится понятным исключительный интерес к этим числам.

К тому же за 6 дней был сотворен мир, а Луна обновляется за 28 суток.

Однако и с математической точки зрения чётные совершенные числа по-своему уникальны. Все они - треугольные.

Это значит, что, взяв совершенное число шаров, всегда можно сложить из них равносторонний треугольник.

Сумма величин, обратных всем делителям числа, включая само число, всегда равна двум.

Остаток от деления совершенного числа, кроме 6, на 9 равен 1.

В двоичной системе совершенное число Pp начинается p единицами, потом следуют p-1 нулей.

P2=110, P3=11100, P5 =111110000,

P7 =1111111000000 и т.д.

Последняя цифра чётного совершенного числа или 6, или 8, причём, если 8, то ей предшествует 2.

Леонард Эйлер доказал, что все чётные совершенные числа имеют вид

2p-1*Mp, где Mp-простое число Мерсенна.

Однако до сих пор не найдено ни одного нечётного совершенного числа.

Высказано предположение (БрайенТакхерман,США), что если такое число

существует, то оно должно иметь не менее 36 знаков.

```
//совершенные числа,не превышающие n
   int i,n,summa,d;
   cout << "введите n\n" ;
   cin >> n;
   for (i=3; i<=n; i++)
   {
     summa=1;
     for (d=2; d <= i/2; d++)
          if (i%d==0)
               summa=summa+d;
     if (i==summa)
          cout << i << endl;
   }
   return 0;
}
   // первых n совершенных чисел
     int  n,p,i,flag,ch;
     n=6;
     p=2;i=1;
     while (i<=n)
     {
          //простое или нет
          int j=2;
          flag=1; //да
          while ((j<=p/2) && (flag))
               if ( !(p%j))
                    flag=0; //нет
               else ++j;
          if ((flag)||(p==2)||(p==3))
          {
               ch=step(2,p-1)*(step(2,p)-1);
               int step=1;
               int m=p-1;
               int x=2;
               while  (m != 0)
               {
                    if (m % 2 == 1) step=step*x;
                    x=x*x;
                    m=m / 2;
               }
```

```
                ch=step*(step*2-1);
                cout << ch << endl;
                i++;
            }
            p++;
    }
    return 0;
}
```

12. Дружественные числа

Это такие два числа, каждое из которых равно сумме делителей второго. Наименьшие из дружественных чисел 220 и 284 были известны еще пифагорейцам, которые считали их символом дружбы. Следующая пара дружественных чисел 17296 и 18416 была открыта французским юристом и математиком Пьером Ферма лишь в 1636 году, а последующие числа находили Декарт, Эйлер и Лежандр. Шестнадцатилетний итальянец Никколо Паганини (тезка знаменитого скрипача) в 1867 году потряс математический мир сообщением о том, что числа 1184 и 1210 дружественные! Эту пару, ближайшую к 220 и 284, проглядели все знаменитые математики, изучавшие дружественные числа.

13. Числа Армстронга

153, 370, 371, 407, ...

n-значное число называется числом Армстронга, если оно равно сумме n-ых степеней своих цифр.

14. Числа Смитта

Число называется числом Смита, если сумма цифр числа равна сумме цифр разложения этого числа на простые множители. Например:

4937775 - число Смита

4937775=3 x 5 x 5 x 65837
Сумма цифр числа - 42
Сумма цифр множителей - 42
Числа Смита: **4, 22, 27, ...**
На интервале (0, 10 000) - 376 чисел Смита
На интервале (0, 100 000) - около 3300 чисел Смита
На интервале (0, 1 000 000) - 29928 чисел Смита
Количество чисел Смита бесконечно.

15. Числа Фибоначчи

Число называется числом Фибоначчи, если оно является одним из членов последовательности:

$$f_n = f_{n-1} + f_{n-2}$$

где

$$f_0 = 1 \text{ и } f_1 = 1.$$

16. Для заданного натурального числа определить количество единичных бит в его представлении

Считаем, что число помещается в 2 байта памяти.

Суть алгоритма в том, что заводится битовая "маска" (переменная mask), содержащая единственный ненулевой бит, позиция

которого при работе цикла меняется от младшего к старшему (либо в обратную сторону).

При единичном значении очередного бита проверяемого числа (переменная ***a***), логическое умножение очередного значения маски на содержимое проверяемого числа маску не меняет.

Этот факт и проверяется для увеличения счетчика ***num***.

```
short  a, mask, i, num;
{mask = 1; //двоичное число
           //0000 0000 0000 0001
num = 0;
for (i = 0; i <=15; i++)
 {
    //проверка битовой 1 в соответствующем разряде
    if ((mask & a) == mask ) num++;
   mask = mask << 1;
 }
} //Количество  единиц  в num
```

Второй вариант решения предыдущей задачи

```
short  a, num;
// num-счетчик битовых единиц
// ВВОД  a;
num = 0;
unsigned short tmp = a;
while (tmp > 0)
 {
  //удаление младшей 1 из переменной tmp
  tmp=tmp-(tmp&(tmp^(tmp-1)));
  num++;
 }
```

17. По заданному натуральному числу (4 байта) получить новое число, переставив младшую цифру исходного числа на место впереди старшей

Например, из входного значения 789 должно получиться 978.

Один из возможных алгоритмов решения состоит из следующих шагов:

1. определяется количество цифр *n_dig* в исходном числе (при *n=789: 3 → n_dig*);

2. исходное число обрезается перед младшим разрядом (*78 → tmp*), а значение младшего разряда запоминается (*9 → digit*);

3. умножением на 10^{n_dig-1} выделенная цифра сдвигается в нужный разряд числа ($9*10^2=900$);

4. к результату добавляется обрезанное значение (*900+78=978*).

```
int tmp, digit, n_dig, n, DT;
n=12345;
tmp = n; n_dig = 0;
```

```
while (tmp > 0)
{
   tmp = tmp / 10;
   n_dig++;
}
tmp = n / 10;
digit = n % 10;
DT = digit * pow(10.0,n_dig - 1)+tmp;
//DT=51234;
```

Задачи для самостоятельного решения

1. В битовом представлении заданного натурального числа перенести младший бит, поставив его перед старшим единичным битом.

2. В битовом представлении заданного натурального числа перенести m младших бит, подставив их перед старшими (m не превосходит длины двоичного представления числа).

3. В битовом представлении заданного натурального числа перенести m младших бит, подставив их перед старшими (m не превосходит длины двоичного представления числа).

18. Перевод чисел в различные системы счисления

Перевести натуральное число *n* в систему счисления с основанием *p*.

Очевидно, в результате нужно получить разложение числа *n* в позиционной нумерации A_p, то есть символьную строку вида $a_k a_{k-1}... a_1 a_0$ (где a_i принадлежит A_p), которая соответствует сумме $n=a_k*p^k+a_{k-1}*p^{k-1}+... +a_1*p^1+a_0$.

В частном случае при $p=10$ мы как раз и имеем "обычное" десятичное разложение.

Программная реализация алгоритма связана с некоторыми ограничениями.

Поскольку в вычислениях применяются операции / и %, используемые в языке программирования для обработки десятичных чисел, да и результат тоже представлен как десятичная запись - символами из A_{10}, ограничимся диапазоном $2 \le p \le 9$.

```
//перевод числа 255 в 8-ую с/с
int a, r;
int n=255, p=8;
a = 0;  r = 1;
while (n >= 1)
  {
   a = a + (n % p) * r;
   r = r * 10;
   n = n / p;
  }
cout<<a<<endl; //a=377
```

КОНЕЦ ЛЕКЦИИ