Алгоритмизация и программирование I

Лекция 8

Повторение

- 1. Что будет выведено на экран?

```
#include <iostream>
using namespace std;
void main()
{
  int a, *p;
  a=3;
  p=&a;
 *p*=a;
  cout << a;
}
```

Повторение

2) Что будет выведено на экран?
3) Что надо исправить для получения правильного результата для всех целых чисел?

```
#include <iostream>
using namespace std;

int sum_c(int m)
{  int s=0;
   while (m!=0)
   {
        s+=m%10;
        m/=10;
   }
  return s;
}
```

```
int main ()
{
  int m,k;
  m=123;
  k=sum_c(m);
  cout<<"k="<<k<<"\n";
  cout<<"m="<<m<<"\n";
  m=-568;
  k=sum_c(m);
  cout<<"k="<<k<<"\n";
  cout<<"m="<<m<<"\n";
return 0;
}
```

ОТВЕТЫ:

1) 9

2) Нахождение суммы цифр целого числа

3) Исправления:k=sum_c(abs(m));
Подключить библиотеку: #include <cmath>

Лекция 8

- Рекурсия

Пример

- Что будет делать следующая программа?

```
#include <iostream>;
using namespace std;
void privet();
{
   cout<<”Привет”;
   privet();
}
void main()
{
privet();
}
```

Примеры рекурсивных определений

- Определение факториала: $n!=1*2*3*...*n$.

$$n! = \begin{cases} 1, & \text{если } n \le 1, \\ (n-1)! \times n, & \text{если } n > 1. \end{cases}$$

- Определение функции $K(n)$, которая возвращает количество цифр в заданном натуральном числе n:

$$K(n) = \begin{cases} 1, & \text{если } n < 10, \\ K(n/10) + 1, & \text{если } n >= 10. \end{cases}$$

- *Задание.* Определите функцию $S(n)$, которая возвращает сумму цифр в заданном натуральном числе n.

Рекурсия

- Рекурсией называется ситуация, когда какой-то алгоритм вызывает себя прямо (прямая рекурсия) или через другие алгоритмы (косвенная рекурсия) в качестве вспомогательного. Сам алгоритм называется рекурсивным.
- Рекурсивное решение задачи состоит из двух этапов:
  - исходная задача сводится к новой задаче, похожей на исходную, но несколько проще;
  - подобная замена продолжается до тех пор, пока задача не станет тривиальной, т. е. очень простой.

Пример

- Что выведет на экран следующая программа, если N=123:

```
#include <iostream>
using namespace std;
void f(int x)
{
  cout<<x % 10<<" ";
  if(x>10) f(x/10);
}
void main()
{   int N;
    cout<<"N=";
    cin>>N;
    f(N);
}
```

Пример

- Как изменить программу, чтобы цифры числа выводились в правильном порядке?

```
#include <iostream>
using namespace std;
void f(int x)
{
  if(x>10) f(x/10);
  cout<<x % 10<<" ";
}
void main()
{   int N;
     cout<<"N=";
     cin>>N;
     f(N);
}
```

Действия, выполняемые на рекурсивном спуске

- Порождение все новых вызовов рекурсивной подпрограммы до выхода на граничное условие называется ***рекурсивным спуском***.
- Максимальное количество вызовов рекурсивной подпрограммы, которое одновременно может находиться в памяти компьютера, называется ***глубиной рекурсии***.

тип rec(параметры)
{
  <*действия на входе в рекурсию***>;**
  If <*проверка условия***> rec(параметры);**
  [else S;]
}

- Обычно проверяют, что с каждым рекурсивным вызовом значение какого-то параметра уменьшается, и это не может продолжаться бесконечно.

Действия, выполняемые на рекурсивном возврате

- Завершение работы рекурсивных подпрограмм, вплоть до самой первой, инициировавшей рекурсивные вызовы, называется ***рекурсивным возвратом***.

тип Rec (параметры);
{
  If <*проверка условия***> Rec (параметры);**
  [else S1];
  <*действия на выходе из рекурсии***>;**
}

Действия, выполняемые на рекурсивном спуске и на рекурсивном возврате

```
тип Rec (параметры);
{
  <действия на входе в рекурсию>;
  If <условие> Rec(параметры);
  <действия на выходе из рекурсии>
}
```

Или

```
тип Rec(параметры);
{
  If <условие>
    {
    <действия на входе в рекурсию>;
    Rec;
    <действия на выходе из рекурсии>
   }
}
```

Задачи, решаемые с помощью рекурсии

- Вычислить факториал (n!), используя рекурсию.
- Вычислить степень, используя рекурсию.
- Вычислить *n*-е число Фиббоначи , используя рекурсию.

Задача 1. Вычисление n!

Вычислить факториал (n!), используя рекурсию.

Исходные данные: n

Результат: n!

- Рассмотрим эту задачу на примере вычисления факториала для n=5. Более простой задачей является вычисление факториала для n=4. Тогда вычисление факториала для n=5 можно записать следующим образом:

5!=4!*5.

Аналогично:

4!=3!*4;

3!=2!*3;

2!=1!*2 ;

1!=0!*1

Тривиальная (простая) задача:

0!=1.

Можно построить следующую *математическую модель*:

$$f(n) = \begin{cases} 1, \text{n} = 0 \\ f(n-1) * n, \text{n} \geq 1 \end{cases}$$

Программа. Вычисление n!

```
#include <iostream.h>
int fact(int n)
{
    if (n==0) return 1;      //тривиальная задача
    return (n*fact(n-1));
}

void main()
{
    cout<<"n?";
    int k;
    cin>>k;  //вводим число для вычисления факториала
    cout<<k<<"!="<<fact(k); //вычисление и вывод результата
}
```

Как работает рекурсия

- Факториал:

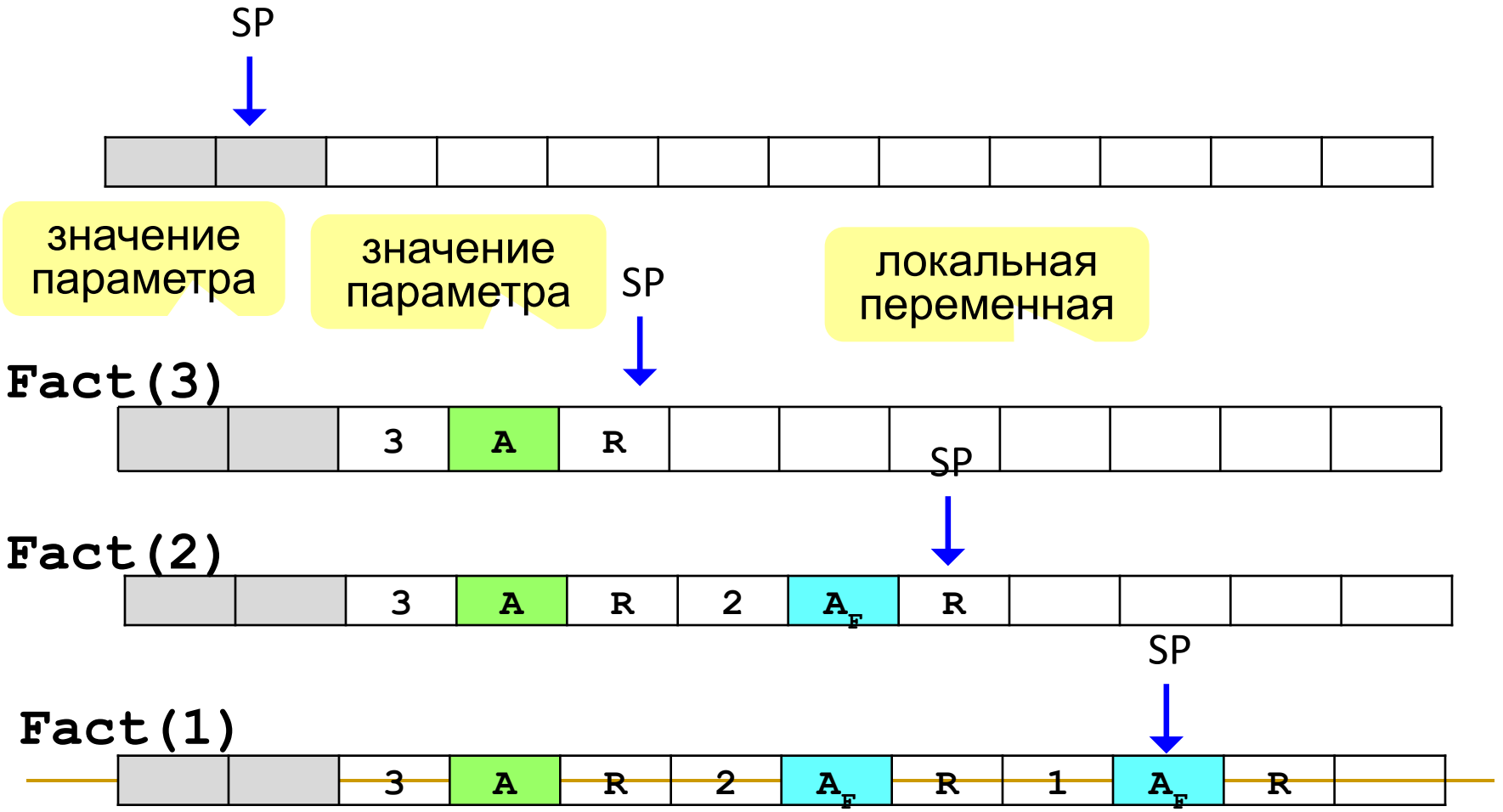
$$N! = \begin{cases} 1, & N = 1 \\ N \cdot (N-1)!, & N > 1 \end{cases}$$

```
int Fact ( int N )
{
   int F;
   cout << "-> N=" << N << endl;
   if ( N == 1 )
     F = 1;
   else F = N * Fact(N - 1);
   cout << "<- N=" << N << endl;
   return F;
}
```

```
-> N = 3
   -> N = 2
      -> N = 1
      <- N = 1
   <- N = 2
<- N = 3
```

Стек

Стек – область памяти, в которой хранятся локальные переменный и адреса возврата.



Рекурсия

- При каждом рекурсивном вызове информация о нем сохраняется в специальной области памяти, называемой ***стеком***.
- В стеке записываются значения локальных переменных, параметров подпрограммы и адрес точки возврата.
- Какой-либо локальной переменной *A* на разных уровнях рекурсии будут соответствовать разные ячейки памяти, которые могут иметь разные значения.
- Воспользоваться значением переменной *A i*-ого уровня можно только находясь на этом уровне.
- Информация в стеке для каждого вызова подпрограммы будет порождаться *до выхода на граничное условие*.
- В случае отсутствия граничного условия, неограниченный рост количества информации в стеке приведёт к аварийному завершению программы за счёт переполнения стека.

Задача 2. Вычисление степени

Исходные данные: x,n

Результат: x^n

Математическая модель:

$$pow(x,y) = \begin{cases} 1, \mathrm{n} = 0 \\ pow(x, n-1) * x, \mathrm{n} \geq 1 \end{cases}$$

Программа. Вычисление x^n

```
#include <iostream.h>
int pow( int x,int n)
{
    if(n==0)return 1;          //тривиальная задача
    return(x*pow(x,n-1));
}
void main()
{
    int x,k;
   cout<<"n?";
    cin>>x;                    //вводим число
    cin>>k;                   //вводим степень
                              //вычисление и вывод результата
   cout<<x<<"^"<<k<<"="<<pow(x,k);
}
```

Задача 3. Вычисление n-го числа Фибоначчи.

- $F_1 = F_2 = 1$
- $F_n = F_{n-1} + F_{n-2}$ при $n > 2$

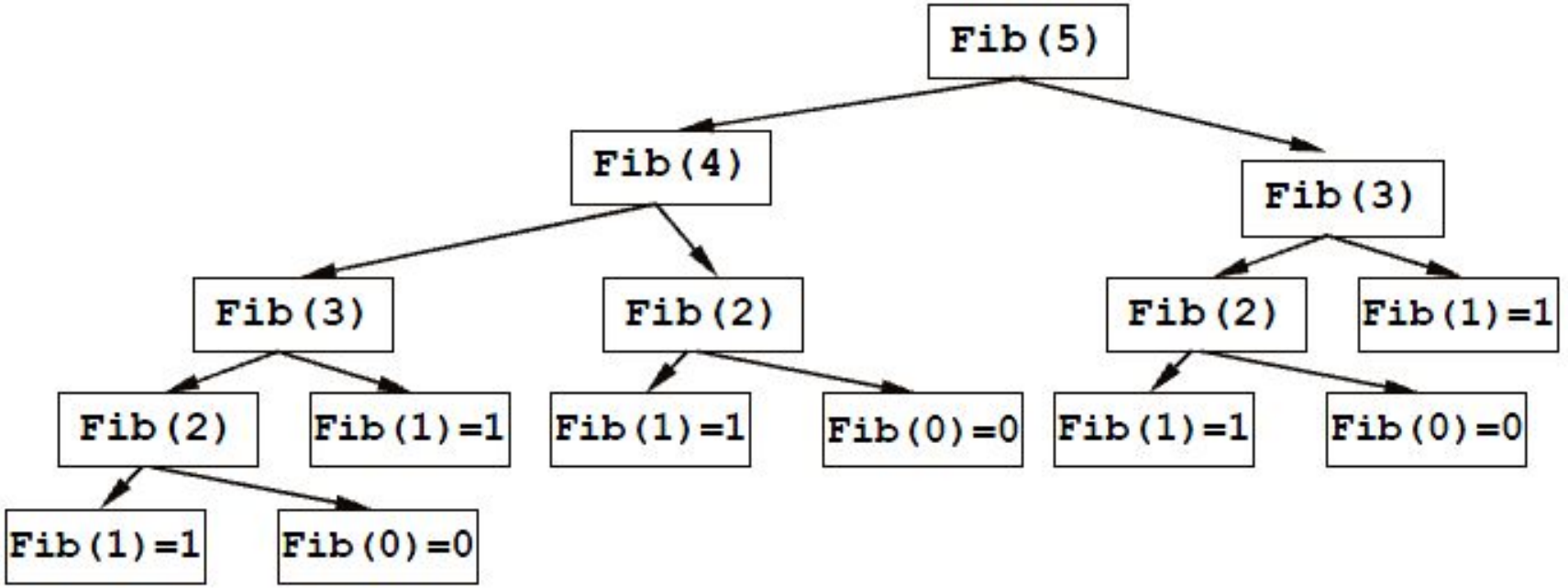
1, 1, 2, 3, 5, 8, 13, 21, 34, …

Программа. Числа Фибоначчи

```
#include <iostream>
using namespace std;
int fib(int x)
{
  if(x==1 || x==2) return 1;
  else return fib(x-1)+fib(x-2);
}
void main()
{   int N;
     cout<<"N=";
     cin>>N;
     cout<<fib(N)<<endl;
}
```

Числа Фибоначчи

- Каждый рекурсивный вызов при **n > 1** порождает еще 2 вызова функции, многие выражения (числа Фибоначчи для малых **n**) вычисляются много раз.



Задача 4. Вычисление суммы цифр числа

```
int sumDig ( int n )
{
   int sum;
  sum = n %10;
  if ( n >= 10 )
   sum += sumDig ( n / 10 );
  return sum;
}
```

Задача 5. Алгоритм Евклида

- **Алгоритм Евклида**. Чтобы найти НОД двух натуральных чисел, нужно вычитать из большего числа меньшее до тех пор, пока меньшее не станет равно нулю. Тогда второе число и есть НОД исходных чисел.

```
int NOD ( int a, int b )
{
  if ( a == 0 || b == 0 )
    return a + b;
  if ( a > b )
      return NOD( a - b, b );
  else return NOD( a, b – a );
}
```

условие окончания рекурсии

рекурсивные вызовы

Задача 6

- Найти сумму 1!+2!+3!+…+n!

Программа

```
#include <iostream>
using namespace std;

int fact(int n)
{
  return (n>1) ? n * fact(n - 1) : 1;
}
int sum(int k)
{
  if (k==0) return 0;
  else return sum(k-1)+fact(k);
}

void main()
{
   int n ;
   cin >> n;
   cout << "Sum="<< sum(n)<< endl;
}
```

Задание 7

- Вычислить сумму S=2+4+6+8+…2*n.

```
#include <iostream>
using namespace std;
int S(int n)
{
    if (n) return (S(n-1)+2*n);
    else return 0;
}
void main()
{
  int n;
  cin>>n;
  cout<<S(n);
}
```

Задание 8

- Определить максимальную цифру целого числа и ее позицию в числе (считать, что цифры пронумерованы справа налево).

```
#include <iostream>
#include <cmath>
using namespace std;
void max_c(int n, int &max, int i, int &i_max)
{
    if (n)
    {
      max_c(n/10, max,i+1,i_max);
      if (max < n %10){max=n%10; i_max=i;}
    }
    else max=0;
}
void main()
{
  int n, m,i=1,im;
  cin>>n;
  max_c(abs(n),m,i,im);
  cout<<m<<" "<<im;
}
```

Задание 9

- Дано натуральное число *N*, заданное в четверичной системе счисления. Написать рекурсивный алгоритм позволяющий перевести это число в десятичную систему счисления.

```
#include <iostream>
#include <cmath>
using namespace std;
int per4(int p4,int t)
{
    if (p4)
      return per4(p4/10,t*4)+p4%10*t;
    else return 0;
}
void main()
{
  int n, t=1;
  cin>>n;
  cout<< per4(n,t);
}
```

Задание 10

- Что будет изображено на экране?

```
#include <iostream>
#include <cmath>
using namespace std;
void print (int m, int n)
{
  int j;
  for(j=1;j<=m;j++) cout<<" ";
  cout<<"*";
  for(j=1;j<=n;j++) cout<<" ";
  cout<<"*";
  for(j=1;j<=n;j++) cout<<" ";
  cout<<"*\n";
}
```

```
void usor(int i)
{
  if (i==4) cout<<"*******\n";
  else
  {
    print(i-1,3-i);
       usor(i+1);
       print(i-1,3-i);
  }
}
void main()
{
  usor(1);
}
```

Косвенная рекурсия

```
#include <iostream>
using namespace std;
int A;
void Rec2 (int& Y);

void Rec1 (int& X)
{
  X= X-1;
    if (X>0)
     {
        cout<<X<<"\n";
        Rec2(X);
     }
}
```

```
void Rec2 (int& Y)
{
  Y= Y /2;
    if (Y>2)
     {
        cout<<Y<<"\n";
        Rec1(Y);
     }
}

void main()
{
    A= 15;
     Rec1(A);
    cout<<A<<"\n";
}
```

Рекурсия – «за» и «против»

- с каждым новым вызовом расходуется память в стеке (возможно переполнение стека)
- затраты на выполнение служебных операций при рекурсивном вызове
- +: программа становится более короткой и понятной
- -: возможно переполнение стека; замедление работы

Итерационный алгоритм

- Любой рекурсивный алгоритм можно заменить нерекурсивным:

```
int Fact ( int N )
{
   int F;
   F = 1;
   for(i = 2;i <= N;i++)
      F = F * i;
   return F;
}
```

Задание

- *Дан рекурсивный алгоритм:*

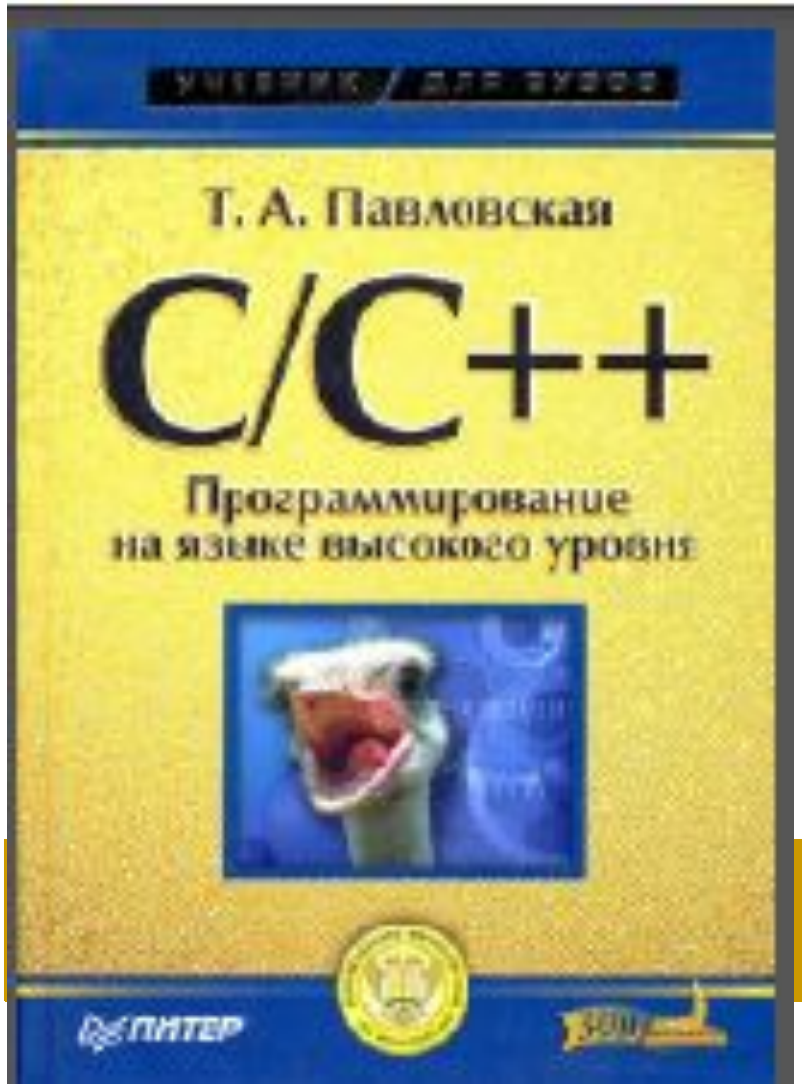
```
#include <iostream>
using namespace std;
void f(int n)
{
  cout <<n<<'\t';
  if (n < 5)
  {
     f(n + 1);
     f(n + 3);
  }
}
void main()
{
   f(1);
}
```

- *Найдите сумму чисел, которые будут выведены при вызове F(1).*

Задание

```
#include <iostream>
using namespace std;
void F(int n)
{
   cout<<'*';
   if (n > 0 )
   {
     F(n-2);
     F(n / 2);
   }
}
void main()
{
     F(5);
}
```

Домашнее задание



1) Стр. 82