Ниже, для понимания механизма выполнения рекурсивных функций, приведены несколько схем демонстрирующих этот механизм. Тексты рассматриваемых функций содержатся в проекте **recurs (Google диск).**

Более подробно схемы рекурсии будут рассмотрены на лекции

-----------------------------------------------------------------------------------------------------------------------------------------------------------

Вначале несколько понятий:

**Куча** – раздел оперативной памяти (ОП) где размещаются динамические переменные.

**Сегмент bss** – (или «неинициализированный сегмент данных»), где хранятся глобальные и статические переменные инициализированные нулем.

**Стек вызовов** – ОП, где хранятся параметры функции, локальные переменные и другая информация, связанная с функциями.

**Переполнение стека**

Стек имеет ограниченный размер и, следовательно, может содержать только ограниченный объем информации. В Windows размер стека по умолчанию составляет 1МБ. На некоторых других Unix-системах этот размер может достигать и 8МБ. При попытке программы поместить в стек слишком много информации (а требуемой для этого памяти уже нет в наличии), это приводит к переполнению стека.

Переполнение стека является результатом добавления слишком большого количества переменных в стек и/или создания слишком большого количества вложенных вызовов функций (например, когда функция A() вызывает функцию B(), которая вызывает функцию C(), а та, в свою очередь, вызывает функцию D() и т.д.). Переполнение стека обычно приводит к сбою в программе, например:

int main()

{

    int stack[1000000000]; // попытка добавить огромный массив в стек вызовов

    return 0;

}

или

void funk()

{

    funk();

}

int main()

{

    funk();

    return 0;

}

В программе, приведенной выше, фрейм стека добавляется в стек каждый раз, когда вызывается функция funk(). **Фрейм** (или **«кадр»**) **стека** отслеживает все данные, связанные с одним вызовом функции Поскольку функция funk() вызывает сама себя бесконечное количество раз, то в конечном итоге в стеке не хватит памяти, что приведет к переполнению стека.

------------------------------------------------------------------------------------------------------------------------------------------------------------

Рекурсивная функция вычисления факториала числа N! = 1 · 2 · 3 · … · N

=========================== Вариант 1 ====================================

fact1(n-1)\*n **Системный стек**

n == 5

**fact(4)** \* n

5! = 120 n

24

**1**

**2**

n == 4

**fact(3)** \* n

``

6

n == 3

**fact(2)** \* n

**3**

2

2

**4**

n == 2

**fact(1)** \* n

**5**

1

2

n == 1

**fact(0)** \* n

int fact1(int n)

{

if (n==1 || n==0) return (1); // выход из рекурсии

return fact1(n-1)\*n;

}

=========================== Вариант 2 ====================================

i\*fact2(n) **Системный стек оперативная память**

n==5, i==0 ( 1 )

**i \* fact(5)**

5! = 120 n i

0 (i++) 1

1 (i++) 2

2 (i++) 3

3 (i++) 4

4 (i++) 5

120

**5**

n==5, i==1 ( 2 )

**i \* fact(5)**

60

**5**

**5**

n==5, i==2 ( 3 )

**i \* fact(5)**

20

**5**

**5**

n==5, i==3 ( 4 )

**i \* fact(5)**

5

n==5, i==4 ( 5 )

**i \* fact(5)**

int fact2(int n)

{

static long i; // i-статическая переменная, чтобы сохранять своё

// последнее значение при каждом новом вызове функции

int factorial;

if(n <= 1) // если число = 0 или 1

return 1; // факториал равен 1

if (++i < n) // используем прекремент, т.е. предварительное

{ // увеличение i на единицу

factorial = i\*fact2(n); // заново вызываем функцию факториала

i--; // декрементируем i

return factorial;

}

i--;

return n;

}

=========================== Вариант 3 ====================================

n\*fact1(n-1) **Системный стек**

n == 5

n \* **fact(4)**

5! = 120 n

24

**1**

**2**

n == 4

n \* **fact(3)**

``

6

n == 3

n \* **fact(2)**

**3**

2

2

**4**

n == 2

n \* **fact(1)**

**5**

1

2

n == 1

n \* **fact(0)**

int fact3(int n)

{

return n ?n\*fact3(n-1):1;

}

Рекурсивная функция выполнения сортировки числового массива методом Хоора.

0 1 2 3 4 5 6 7 8 9 индекс

**l (las) r**

**7 5 2 8 6 3 4 8 1 5** las=0

**6 5 2 8 7 3 4 8 1 5** las=1, 2

**6 5 2 3 4 8 7 8 1 5** las=3, 4

**6 5 2 3 4 1 5 8 8 7** las=5, 6

**6 5 2 3 4 1 5 8 8 7** a[l==0] и a[las==6] меняем местами

**5 5 2 3 4 1 6 8 8 7** hoor(a, 0, 5); //1

**l (las) r**

**5 5 2 3 4 1 6 8 8 7** las=0

**2 5 5 3 4 1 6 8 8 7** las=0

**2 1 5 3 4 5 6 8 8 7** las=1

**1 2 5 3 4 5 6 8 8 7** a[l==0] и a[las==1] меняем местами

**1 2 5 3 4 5 6 8 8 7** hoor(a, 2, 5); //2

**l (las) r**

**1 2 5 3 4 5 6 8 8 7** las=2

**1 2 3 5 4 5 6 8 8 7** las=2

**1 2 3 5 4 5 6 8 8 7** a[l==2] и a[las==2] меняем местами

**1 2 3 5 4 5 6 8 8 7** hoor(a, 3, 5); //2

**l (las) r**

**1 2 3 5 4 5 6 8 8 7** las=3

**1 2 3 4 5 5 6 8 8 7** las=3

**1 2 3 4 5 5 6 8 8 7** a[l==3] и a[las==3] меняем местами

**1 2 3 4 5 5 6 8 8 7** hoor(a, 4, 5); //2

**1 2 3 4 5 5 6 8 8 7** las=4

**1 2 3 4 5 5 6 8 8 7** a[l==4] и a[las==4] меняем местами

**1 2 3 4 5 5 6 8 8 7** hoor(a, 5, 5); //2

void hoor(int \*a, int l, int r)

{

int i, las;

if (l >= r) return;

swap(a, l, (l + r) / 2); //делящий эл-т переносится в a[l] (a[l]<->a[(l+r)/2])

las = l; //позиция посл-го эл-та большего чем делящий

for (i = l + 1; i <= r; i++) //деление [l,r] на [l,las-1] и [las+1,r]

if (a[i]<a[l]) // ЦИКЛ УБРАТЬ !!!!!!!!!!!!!!!!!!!!

swap(a, ++las, i);

swap(a, l, las);

hoor(a, l, las - 1); // 1 //сортировка для [l,las-1]

hoor(a, las + 1, r); // 2 //сортировка для [las+1,r]

}

0 1 2 3 4 5 6 7 8 **dl==9** индекс в строке Возврат информации

**i Стек** из стека в строку

**I**

**dl-8**

**dl-7**

**dl-6**

**dl-5**

**dl-4**

**dl-3**

**dl-2**

**dl-1**

**dl-0**

**I**

**8**

**7**

**6**

**5**

**4**

**3**

**2**

**1**

**0**

**a b c d e f g h k** \0

**k**

**dl-i**

**k h g f e d c b a** \0

**h 5**

**g 5**

**f 5**

**e 5**

void revers1(char \*st, int dl)

{

static int i;

**d 5**

char c;

c = \*(st + i++);

if (c) revers1(st, dl);

**c 5**

else return;

\*(st + dl - (--i)) = c;

}

**b**

**a5**

Занесение инфор-

мации из строки

в стек