Рекурсия,рекурсия,рекурсия,рекурсия,рекурсия,рекурсия,рекурсия,рекурсия,рекурсия,рекурсия, рекурсия,рекурсия,рекурсия,рекурсия,рекурсия,рекурсия,рекурсия,рекурсия,рекурсия, рекурсия, рекурсия,рекурсия,рекурсия,рекурсия,рекурсия,рекурсия,рекурсия,рекурсия,рекурсия, рекурсия, рекурсия,рекурсия,рекурсия,рекурсия,рекурсия,рекурсия,рекурсия,рекурсия,рекурсия, рекурсия.

Стоп, рекурсия!

Как ты уже догадался сегодня будем разбирать такую клёвую штуку как рекурсия.

Так что такое рекурсия? Давай заглянем в википедию!

Реку́рсия — определение, описание, изображение какого-либо объекта или процесса внутри самого этого объекта или процесса, то есть ситуация, когда объект является частью самого себя. Термин «рекурсия» используется в различных специальных областях знаний — от лингвистики до логики, но наиболее широкое применение находит в математике и информатике.

Кто-нибудь понял что тут вообще написано? Я вот лично не очень, поэтому попробую сформулировать понятие рекурсии для программирования своими словами.

Рекурсия – это когда функция вызывает саму себя.

И все?Рекурсия это просто самовызов функции внутри себя? Ну изи тогда, пойду и сдам экзамен на 10.Вот чтобы вся прога была такой легкой!

Постой,студент. Определение ты запомнил, а вот понял ли ты вообще что такое рекурсия, как с ней работать и зачем она нужна вообще?

Вот тебе простой вопрос для разогрева. Является ли данная функция рекурсивной?

Void fun()

{

fun();  
}

Ну конечно ДА! Ты за дураков нас держишь что ли? Функция fun все время вызывает сама себя. Это – рекурсия! Я же помню определение!

Верно – это рекурсия. А вот такая функция?

Void fun()

{

fun2();  
}

Void fun2()

{

fun();  
}

Хмм,тут вроде бы рекурсия , но как-то не понятно.Функция fun вызывает же не саму себя , она вызывает fun2().

Верно подмечено.Функция fun вызывает fun2 , но fun2 вызывает fun , поэтому эта тоже будет рекурсией, но не простой , а косвенной!

Бывает 2 типа рекурсии. Прямая и косвенная рекурсия.Прямая – это когда функция вызывает саму себя. Это был первый пример:

Void fun()

{

fun();  
}

А косвенная рекурсия – это когда функция вызывает саму себя через другую функцию.

Void fun()

{

fun2();  
}

Void fun2()

{

fun();  
}

Здесь функция fun вызывает саму себя через функцию fun2.

Ну, с тем какие рекурсии бывают разобрались , а зачем они нужны вообще?

Хм,ну давай те решим одну задачку. Нам нужно n раз вывести некое сообщение. Напишу ка я рекурсивную функцию для этого.

Void fun(int a)

{

if(a!=0)

{

Cout<<”Здесь могла бы быть ваша реклама!”<<endl;

fun(--a);

}

}

Void main()

{

fun(10);

}

Теперь в консоль выведется 10 раз фраза «Здесь могла бы быть ваша реклама!».

Хех, а зачем тут вообще рекурсивную функцию делать? Я же циклы знаю и магу написать программу для вывода n сообщений вообще без функций используя только цикл!

Void main()

{

For(int i=0;i<10;i++)

{

Cout<<”Ну купите кто нибудь это место под рекламу!”<<endl;  
}

}

Вот , кода в 2 раза меньше , а результат тот же. Автор, ты не вывозишь. Объясняй нормально в чем прикол рекурсий!

Верно, нужно еще и уметь определять когда нам нужно использовать эту рекурсию. В данной задаче действительно выгодней было использовать цикл for , а не рекурсию.

А теперь давай решим такую задачу. Нужно найти двойной факториал числа n;

Напишем реализацию для цикла for

Void main

{

Int a=1;

If(a%2==0)

{

For(int i=2;i<=n;i+=2)

{

a\*=i;  
}

}

Else

{

For(int i=1;i<=n;i+=2)

{

a\*=i;  
}

}

}

Как-то не очень красиво. Сначала мы определяем четный или не четный будет двойной факториал , а потом пишем цикл специально для нечетных или четных i.А как это будет в виде рекурсии выглядеть?

А выглядеть это будет вот так:

int factarial(int a)

{

if(a == 1)

return 1;

if(a == 2)

return 2;

return a \* factarial(a - 2);

}

void main()

{

cout<<factarial(6);

cout<<factarial(5);

}

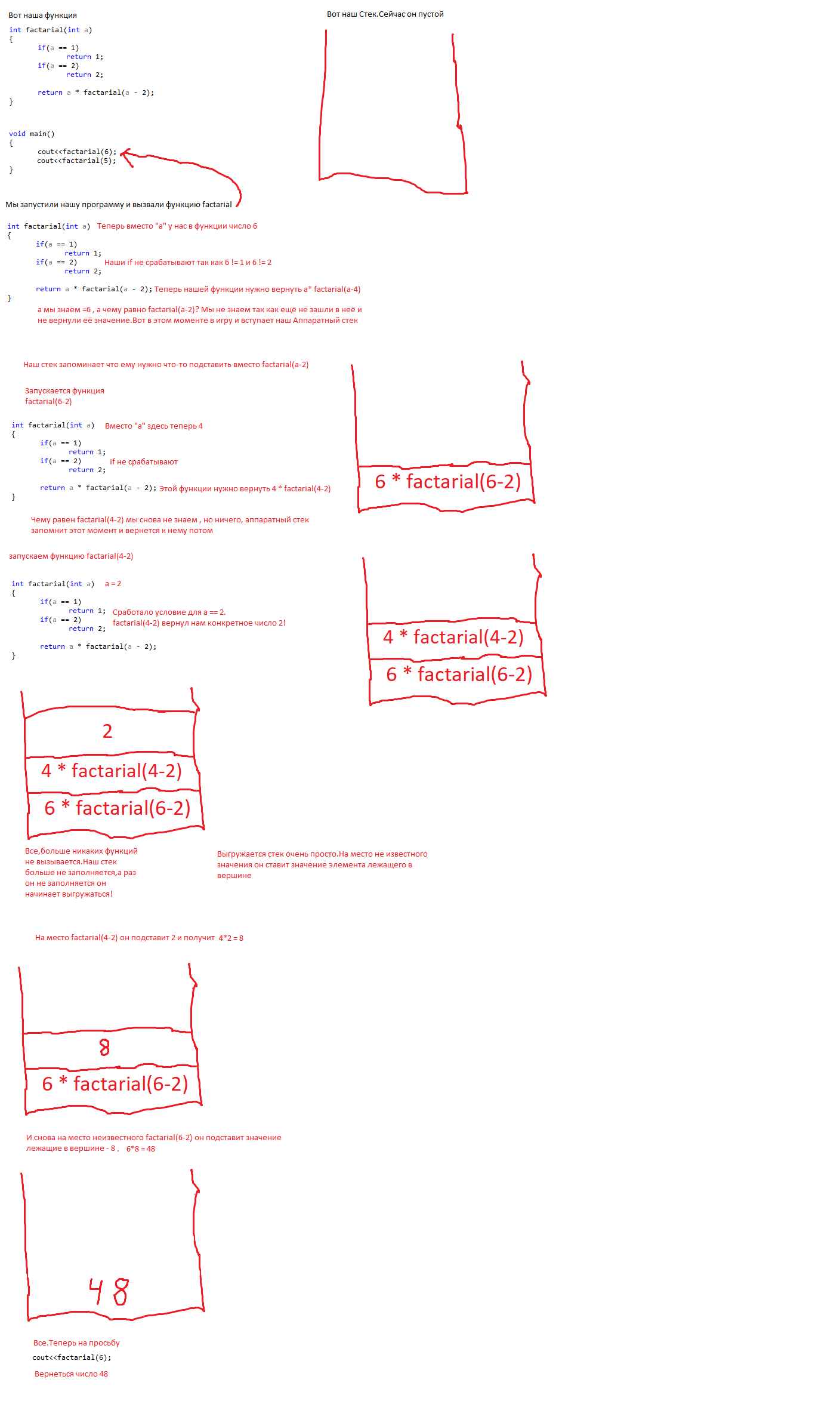
Ого,ну этот способ куда круче и меньше по коду. Но вот только почему это работает? Что за колдунство?

Никакого колдунства. Все в рамках закона и работы со стеком!

Стек?! А причем здесь эта штука? Мы его вроде бы нигде не создавали.

Конечно не создавали , маленькие вы ещё стек настоящий создавать! Поэтому его давно создали за вас и назвали его АППАРАТНЫМ СТЕКОМ! Аппаратный стек ничем не отличается от того что вы учили кроме того, что он умеет ещё с функциями работать и их возвращаемыми значениями.

Для лучшего понимания изобразим работу аппаратного стека графически на примере его работы с функцией factarial которую мы рассматривали выше. Так как рисование в word в школе я прогуливал, то изображу весь процент в paint , а потом вставлю картинку сюда.



Теперь попробуй проделать такие же мысленные операции для cout<<factarial(5);

Ну, подведем итог.Рекурсии нам нужны для написание более красивого кода,а иногда даже более оптимизированного. Мы разобрали как именно работает рекурсия и как ей сильно в этом помагает аппаратный стек. Единственный вопрос на который я не ответил – это когда использовать рекурсию, а когда простые циклы и прочее? Честно, не знаю. Всем пока.

Вот так вот! Проще всего сказать не знаю. Объясняй давай , автор!

Ладно,ладно.Попробую объяснить почему не знаю.Самая главная причина это то что любую рекурсию можно переписать в виде алгоритма без рекурсии.Программист сам решает как ему написать решение для своей задачи.Если ему легче считать числа Фибаначи через рекурсию , то это его выбор.Это просто его способ решения задачи. Главное чтобы ответ получался верным, а получен он с помощью рекурсии или алгоритма это на мой взгляд так же неважно как вести споры о том как лучше получить число 12. Записать выражение в виде 4+8=12 или 8+4=12.

На этом у меня все.Учите прогу,пригодится если ни на одну нормальную работу не возьмут ☺.