Санкт-Петербургский государственный университет
телекоммуникаций им. проф. М.А. Бонч-Бруевича

Рекурсия в C++

СПб ГУТ)))

Функция, которая вызывает сама себя, называется ***рекурсивной функцией***.

Рекурсия - вызов функции из самой функции.

Пример рекурсивной функции - функция вычисления факториала.

Функция factr(), вычисляет факториал целого числа. Факториал числа N является произведением чисел от 1 до N. Например, факториал 3 равен 1*2*3 или 6. Рекурсивная функция factr() и её итеративный эквивалент fact () показаны ниже:

```
5   /* Вычисление факториала числа */
6
7   int factr(int n) /* рекурсивно */
8   {
9   int answer;
10  if(n==1) return(1);
11  answer = factr(n-1)*n;
12
13  return(answer);
14  }
```

```
16  /* Вычисление факториала числа */
17
18  int fact (int n)     /* нерекурсивно */
19  {
20  int t, answer;
21  answer =1;
22  for(t=1; t<=n; t++)
23  answer=answer*(t);
24  return(answer);
25  }
```

Вызов функций и результаты работы

```
26  int main(int argc, char** argv) {
27
28      cout<<"Factorial ne rec "<<fact(5)<<endl;
29      cout<<"Factorial    rec "<<factr(5)<<endl;
30
31      return 0;
32  }
```

fact

Вывод

fact (Собрать, Выполнение) | fact (Выполнение)

```
Factorial ne rec 120
Factorial    rec 120

ВЫПОЛНЕНИЕ SUCCESSFUL (общее время: 140ms)
```

Действие нерекурсивной версии fact() совершенно очевидно. Она использует цикл, начиная с 1 и заканчивая указанным числом, последовательно перемножая каждое число на ранее полученное произведение.

Действие рекурсивной функции factr() более сложно. Когда factr() вызывается с аргументом 1, функция возвращает 1. В противном случае она возвращает произведение factr(n- 1) * n. Для вычисления этого значения factr() вызывается с n-1. Это происходит, пока n не станет равно 1.

При вычислении факториала числа 2, первый вызов factr() приводит ко второму вызову с аргументом 1. Данный вызов возвращает 1, после чего результат умножается на 2 (исходное значение n). Ответ, таким образом, будет 2. Можно попробовать вставить printf() в factr() для демонстрации уровней и промежуточных ответов каждого вызова.

Когда функция вызывает сама себя, в стеке выделяется место для новых локальных переменных и параметров. Код функции работает с данными переменными. Рекурсивный вызов не создает новую копию функции. Новыми являются только аргументы. Поскольку каждая рекурсивно вызванная функция завершает работу, то старые локальные переменные и параметры удаляются из стека и выполнение продолжается с точки, в которой было обращение внутри этой же функции. Рекурсивные функции вкладываются одна в другую как элементы подзорной трубы.

Рекурсивные версии большинства подпрограмм могут выполняться немного медленнее, чем их итеративные эквиваленты, поскольку к необходимым действиям добавляются вызовы функций. Но в большинстве случаев это не имеет значения. Много рекурсивных вызовов в функции может привести к переполнению стека. Поскольку местом для хранения параметров и локальных переменных функции является стек и каждый новый вызов создает новую копию переменных, пространство стека может исчерпаться. Если это произойдет, то возникнет ошибка - переполнение стека.

Основным преимуществом применения рекурсивных функций является использование их для более простого создания версии некоторых алгоритмов по сравнению с итеративными эквивалентами. Например, сортирующий алгоритм Quicksort достаточно трудно реализовать итеративным способом. Некоторые проблемы, особенно связанные с искусственным интеллектом, также используют рекурсивные алгоритмы. Наконец, некоторым людям кажется, что думать рекурсивно гораздо легче, чем итеративно.

При написании рекурсивных функций следует иметь оператор if, чтобы заставить функцию вернуться без рекурсивного вызова. Если это не сделать, то, однажды вызвав функцию, выйти из нее будет невозможно. Это наиболее типичная ошибка, связанная с написанием рекурсивных функций. Надо использовать при разработке функции printf() и getchar(), чтобы можно было узнать, что происходит, и прекратить выполнение в случае обнаружения ошибки.