Интернет-журнал «Мир науки» / World of Science. Pedagogy and psychology https://mir-nauki.com

2018, №4, Том 6 / 2018, No 4, Vol 6 https://mir-nauki.com/issue-4-2018.html

URL статьи: https://mir-nauki.com/PDF/45PDMN418.pdf

Статья поступила в редакцию 20.07.2018; опубликована 07.09.2018

Ссылка для цитирования этой статьи:
Пирогов В.Ю., Баландина И.В. Основы методики обучения рекурсивному программированию // Интернет-журнал «Мир науки», 2018 №4, https://mir-nauki.com/PDF/45PDMN418.pdf (доступ свободный). Загл. с экрана. Яз. рус., англ.

For citation:
Pirogov V.Ju., Balandina I.V. (2018). Foundations of teaching methods of recursive programming. *World of Science. Pedagogy and psychology*, [online] 4(6). Available at: https://mir-nauki.com/PDF/45PDMN418.pdf (in Russian)

УДК 37

Пирогов Владислав Юрьевич
ФГБОУ ВО «Шадринский государственный педагогический университет», Шадринск, Россия
Заведующий кафедрой «Программирования и автоматизации бизнес-процессов»
Кандидат физико-математических наук, доцент
E-mail: Vladislav-133@yandex.ru

Баландина Ирина Викторовна
ФГБОУ ВО «Шадринский государственный педагогический университет», Шадринск, Россия
Доцент кафедры «Программирования и автоматизации бизнес-процессов»
Кандидат педагогических наук
E-mail: piv_vip@mail.ru

Основы методики обучения рекурсивному программированию

Аннотация. В статье рассматриваются вопросы обучения программированию. В частности, исследуется элементы методики преподавания рекурсивного программирования. Авторы предлагают выделить рекурсивное программирование в самостоятельную тему и рассматривать ее наряду с циклическими алгоритмами и процедурным программированием. В статье подробно разбираются виды рекурсии и типовые программные структуры. Авторы, наряду с простой и косвенной рекурсией, также выделяют смешанную рекурсию. Показано, что обычный циклический алгоритм может быть представлен в рекурсивном виде. Предлагается рассматривать рекурсивный алгоритм как обобщение циклического алгоритма. Далее в статье приводятся примеры типовых рекурсивных алгоритмов, в частности рассматривается задача быстрой сортировки. Все представленные в работе авторские положения иллюстрируются программным кодом на алгоритмическом языке С с подробными комментариями. В конце статьи дается несколько общих рекомендаций для обучения рекурсивному программированию:

1. Тема «Рекурсивное программирование» должна быть выделена в общем курсе обучения программированию.
2. Рекурсивное программирование следует рассматривать как технологию, обобщающую обычные циклические алгоритмы.
3. Понятие рекурсии легче всего ввести рассматривая простые задачи, решаемые циклическим перебором. Проводимая аналогия помогает понять рекурсивные алгоритмы.

4. Косвенную рекурсию, в частности, можно рассматривать как обобщение вложенных циклов.
5. На занятиях по рекурсивному программированию важно рассмотреть вопрос экономии памяти и роль в этом глобальных и локальных переменных, а также параметров.
6. Рекурсивный подход позволяет легко решать задачи, которые трудно поддаются другим способам решения.

Ключевые слова: программирование; методика преподавания информатики; рекурсия; информатика; рекурсивное программирование; процедурное программирование; быстрая сортировка

Ведение

Одной из важнейших составляющих содержания такого предмета, как Информатика, в школе является программирование. Эта техническая дисциплина стала возвращаться в школьные программы в 2000-е годы, после периода почти десятилетнего забвения, по мере осознания важности этой отрасли для экономики России. В настоящее время существует довольно много не решенных вопросов, касающихся разных аспектов преподавания программирования: доля часов, отводимых на программирование по отношению к другим темам, содержательная сторона преподавания – какие вопросы нужно включать в школьный курс, какие алгоритмические языки использовать и на базе каких инструментальных средств преподавать. Важной проблемой является и подготовка будущих учителей информатики и переподготовка по вопросам программирования уже сложившихся специалистов.

Вопросу, который будет рассмотрен в статье, обычно уделяется мало внимание, если уделяется вообще. Это связано с его относительной сложностью. Речь идет о такой технологии, как рекурсия или рекурсивное программирование [1]. Очень часто с рекурсией знакомятся при углубленном изучении Информатики и при подготовке к олимпиадам. Но, как правило, рекурсия не рассматривается как самостоятельная технология, позволяющая решить многие задачи, которые обычными способами перебора, решить довольно сложно. Мы рассмотрим в статье некоторые подходы, позволяющие посмотреть на рекурсию как на мощный инструмент, с помощью которого можно решать самые разные задачи.

В действительности сам рекурсивный принцип выходит за рамки разработки программного обеспечения [2]. Он предполагает общий метод определения некого множества или последовательности объектов, основанный на подобии, схожести этих объектов. Метод определения заключается в предположении, что у такой последовательности всегда есть начало, первый элемент. В программировании этой первый вызов рекурсивной процедуры. Вторая половина метода заключается в том, что дается определение произвольного элемента на основе предыдущего. Типичным примером такого подхода является определение предков человека, которое можно сформулировать в виде двух пунктов:

1. Родители человека являются его предками.
2. Предки предков человека являются его предками.

Как видим, данное определение несет в себе алгоритм, позволяющий выделить среди всех людей множество, которое определенным образом связано с данным человеком. Обратим внимание на одну очень важную деталь. У человека два родителя. Следовательно, для исследования своих предков мы должны пройти по двух веткам: материнской и отцовской линии. И так на каждом уровне. Простым циклическим перебором такую задачу решить

довольно сложно, особенно если мы в общем случае не знаем глубину, до которой необходимо провести анализ.

Ниже в статье мы излагаем некоторые подходы, позволяющие упростить изложение материала «Рекурсивное программирование» в школе и Вузе на занятиях по программированию.

О процедурном программировании

Прежде чем говорить о рекурсии необходимо ввести такое фундаментальное понятие, как процедура и процедурное программирование [3, 4, 5]. Это понятие настолько важно, что невозможно говорить об обучении написанию программ, без усвоения процедурного подхода, без использования процедур при решении задач программирования. Исторически в разных языках программирования кроме термина «процедура», используется термин «функция». Например, в языке Паскаль есть и процедуры, и функции. В языке С используется только термин «функция». В сущности, везде речь идет об одном и том же. В дальнейшем будем использовать термины «процедура» и «функция» как синонимы.

Три важнейших аспекта должны быть усвоены обучаемыми в первую очередь:

1. Использование процедур позволяет заменять подобные фрагменты программы на вызов процедуры и тем самым сократить исходный код.
2. Процедурный подход делает программу более понятной и легко читаемой.
3. Использование параметров значительно расширяет возможности процедурного подхода. Примером параметризации может служить решение квадратного уравнения вида $ax^2 + bt + c = 0$, где коэффициенты **a, b, c** используются в качестве параметров, передаваемых в процедуру, вычисляющую корни квадратного уравнения.

Результаты работы процедуры, если они выражаются некоторыми значениями, должны быть возвращены коду, вызвавшему процедуру (например, корни квадратного уравнения). В любом случае такой возврат осуществляется через некоторую область памяти, доступ к которой имеется из кода процедуры, и из кода, откуда осуществляется вызов. На низком уровне это могут быть регистры процессора или некоторая область оперативной памяти [6]. На уровне алгоритмического языка программирования обычно используют параметры, возвращающие значения или глобальные переменные. В действительности же, параметры, возвращающие значение, реализуются через глобальные же переменные (адреса).

Одна из особенностей процедурного подхода заключается в том, что из одной процедуры может быть вызвана другая процедура. В результате в программе может образоваться целая иерархия вызывающих друг друга процедур. На рисунке 1 представлен пример построения программы по процедурному типу. Прямоугольниками на рисунке изображены процедуры, из которых состоит программа. Горизонтальные стрелки в процедурах показывают ход выполнения кода. Изогнутые стрелки показывают вызовы процедур и возврат из них. Хороший стиль программирования требует, чтобы была не только одна точка входа в процедуру, но и одна точка выхода. Как мы увидим в дальнейшем, для рекурсивных алгоритмов это может быть особенно важно.

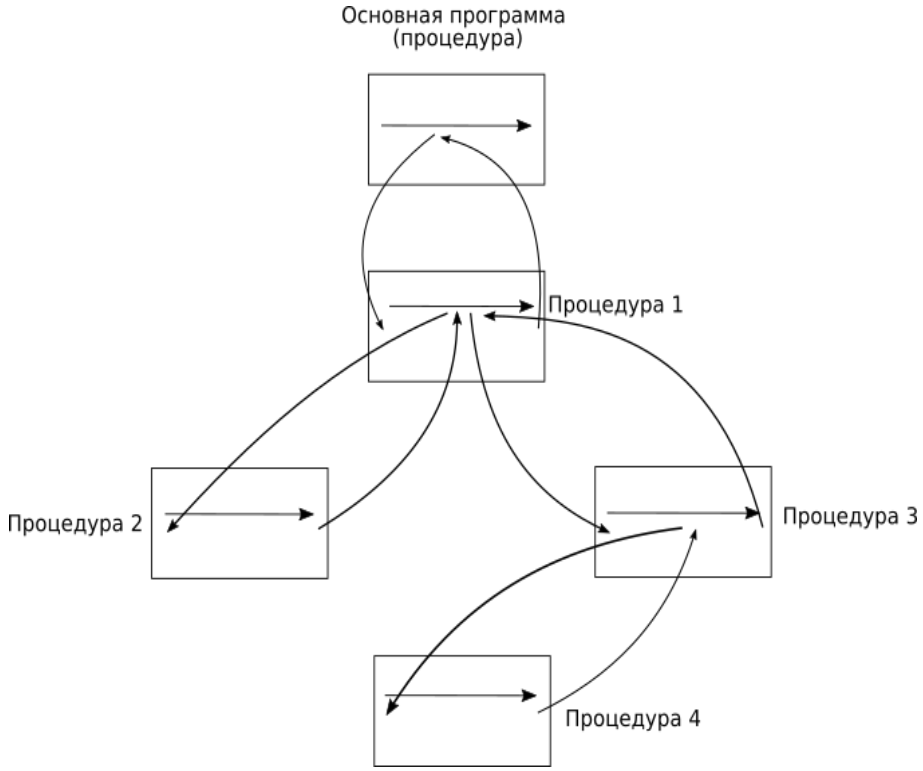

Рисунок 1. *Схема, иллюстрирующая основу процедурного подхода (составлено авторами)*

При вызове процедуры используется стековая память, где, как минимум, хранится, адрес возврата – адрес команды, следующей за вызовом. Через стековую же память передаются в процедуру и параметры. В компилирующих языках для более быстрой передачи параметров используются также регистры процессора, но количество их ограничено и при большом количестве параметров все равно используется стековая передача. Также стековая память используется для локальных переменных, и хранения некоторой другой служебной информации [6]. Это означает, что последовательный вызов одной процедуры из другой приводит к некоторому дополнительному потреблению стековой памяти, которая, естественно, не безгранична.

В листинге 1, представлена программа, демонстрирующая схему, представленную на рисунке 1. Программа не несет в себе никакой особой полезной нагрузки, кроме показа сущности процедурного подхода.

Листинг 1. Пример программы, демонстрирующий схему, представленную на рисунке 1 (язык программирования С).

```
#include <stdio.h>
//прототипы
void prog1();
void prog2();
void prog3();
void prog4();
int main(){
    prog1();
    return 0;
```

```
};
void prog1(){
        printf("Процедура 1\n");
        prog2();
        prog3();
        return;
}
void prog2(){
        printf("Процедура 2\n");
        return;
}
void prog3(){....
        printf("Процедура 3\n");
        prog4();
        return;
}
void prog4(){
        printf("Процедура 4\n");
        return;
}
```

Рекурсия и циклические алгоритмы

Теоретические основы рекурсии можно найти в книгах [7, 8], также укажем на очень хорошее введение в рекурсивное программирование для программистов [9]. Для того, чтобы объяснить, как работает рекурсия, необходимо начинать с очень простых примеров, а главное провести аналогию между циклическим и рекурсивным подходами. Рассмотрим простой перебор значений целых чисел в заданном промежутке. С использованием оператора цикла на языке С это будет выглядеть, например, так for (int i=0; i<100; i++) printf("%d\n",i); – строка программы выводит на консоль последовательно целые числа в промежутке от 0 до 99. Теперь рассмотрим, как этот же самый алгоритм можно реализовать с помощью простого рекурсивного алгоритма. В листинге 2 представлена такая программа.

Листинг 2. Пример рекурсивного алгоритма, выполняющего роль обычного цикла (язык программирования С).

```
include <stdio.h>
//прототип рекурсивной функции
void rec(int,int);
int main(){
        rec(0,100);
        return 0;
```

```
};
//рекурсивная функция, выполняющая роль обычного цикла
void rec(int i, int n1){
        if(i<n1){
                printf("%d\n",i);
                rec(++i,n1);
        }
        return;
}
```

Программа, представленная в листинге 1, содержит такой элемент, как рекурсивная функция. В программе она называется rec. Параметры i и n1 являются аналогами параметра цикла (i) и верхней границы для значений параметра цикла. Каждый вызов функции rec соответствует одной итерации цикла. Первый вызов функции rec(0,100) соответствует первому входу в цикл. Сложнее всего понять, как осуществляется возврат из цепочки рекурсивных вызовов. Но здесь на помощь придет пример, на основе которого обычно объясняется работа стека – стопка тарелок. Вызов рекурсивной функции – одна тарелка кладется в стопку, выход из рекурсии – одна тарелка удаляется из стопки. В сущности, в такой аналогии нет ничего удивительного, ведь вызов процедур основывается на использовании стекового механизма.

Вообще важно дать понять обучаемому, что рекурсивный алгоритм шире простого цикла. Связано это как раз тем, что в рекурсии есть шаг вперед – итерация и шаг назад – возврат из рекурсии. В рекурсивной функции из листинга 2 шаг назад осуществляется, если условие i<n1 не выполняется. В данном примере это только команда return и более ничего, но при необходимости перед этой командой могут выполняться и другие действия. При чем действия любые, в том числе и другой рекурсивный или обычный вызов функции. На данном этапе это, пожалуй, самый важный момент на который следует обратить внимание обучаемых. Он является основным принципиальным отличием рекурсивного алгоритма от циклического – по сути обобщением циклического алгоритма.

Следует также обратить внимание на выход из рекурсивной процедуры. Как видно из листинга 2 рекурсивная процедура имеет одну точку выхода. Данный пример довольно прост, в действительности, однако, рекурсивная процедура может иметь довольно сложную структуру и наличие в ней нескольких точек выхода может привести к тому, что анализ работы алгоритма может оказаться весьма затруднительным.

Типы рекурсии

В предыдущем примере (см. листинг 2) мы имеем образец, так называемой простой рекурсии, когда процедура, называемая рекурсивной, обращается к самой себе. На рисунке 2 представлена схема, по которой работает простая рекурсия. Горизонтальные линии со стрелкой показывают направление выполнения программы. Изогнутые стрелки имеют отношение к вызовам рекурсивной процедуры и возврату из рекурсии. 1 – первый вызов рекурсивной процедуры из основной программы, 2 – возврат из рекурсивной процедуры в основную программу, 3 – рекурсивный вызов, 4 – возврат из рекурсии.

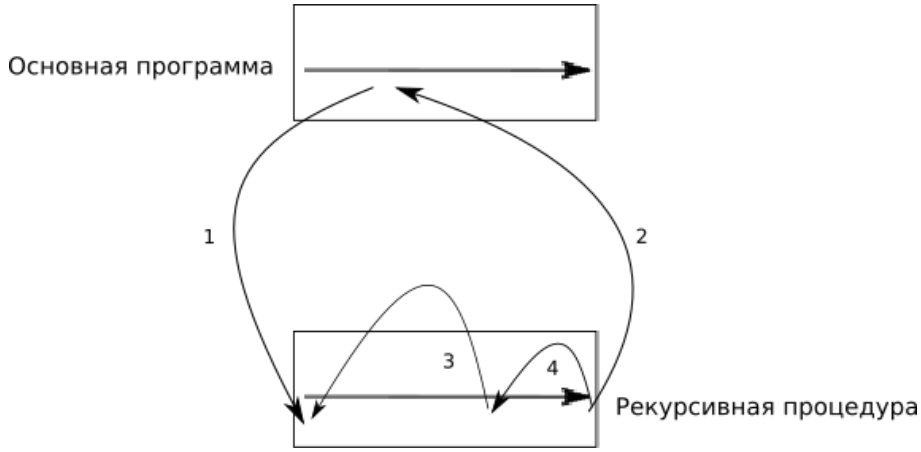

Рисунок 2. *Простая рекурсия (составлено авторами)*

В листинге 3 представлена программа вычисления суммы элементов массива. Это пример простой рекурсии, когда из рекурсивной процедуры происходит обращение к самой себе. Вычисление, которое содержится в алгоритме, в программе, имеет одну важную особенность. Обратим внимание на элемент a[i]+rec1(i+1). Очевидно, что вычисление при i<n не возможно – нужно в начале узнать rec1(i+1). Это вычисление начинается после того, как i достигает значение n и осуществляется начиная с последнего элемента массива – a[n-1].

Листинг 3. Пример рекурсивной программы вычисления суммы элементов массива (язык С).

```
//сумма элементов массива
#include <stdio.h>
int rec1(int);
int a[100];
int n,k;
int main(){
//блок ввода данных из внешнего потока
//размер массива
if(scanf("%d",&n)==EOF){
return 1;
}
//ввод массива
for(k=0; k<n; k++){
if(scanf("%d",&a[k])==EOF){
        return 1;
}
}
//вызов рекурсивной функции
printf("%d\n",rec1(0));
```


45PDMN418
Издательство «Мир науки» \ Publishing company «World of science» http://izd-mn.com

```
//конец программы
 return 0;
};
//рекурсивная функция
int rec1(int i){
if(i<n)
        return a[i]+rec1(i+1);
else return 0;
}
```

Более сложной является косвенная рекурсия. Пусть имеется три процедуры A, B, C. Если вызов процедур осуществляется, например, так: A вызывает B, B вызывает C, C вызывает A, то на лицо косвенная рекурсия. Количество промежуточных звеньев в вызове рекурсивной процедуры A может быть в действительности произвольным. Кроме того, отдельные процедуры, задействованные в такой цепочки вызовов, могут вызывать самих себя, т. е. реализовывать простую рекурсию. В целом же такую схему можно назвать смешанной рекурсией.

На рисунке 3 представлен пример смешанной рекурсии. Из основной программы вызывается процедура 1 (стрелка 1, возврат стрелка 2). Далее из нее вызывается процедура 2 (стрелка 3, возврат стрелка 4). Сама процедура 2 также в некоторой точке вызывает процедуру 1 (стрелка 5, возврат стрелка 6). Имеем, таким образом, косвенную рекурсию. Но из процедуры 2 есть и другой вызов (стрелка 7, возврат – стрелка 8) – самой себя. В данном случае это прямая рекурсия.

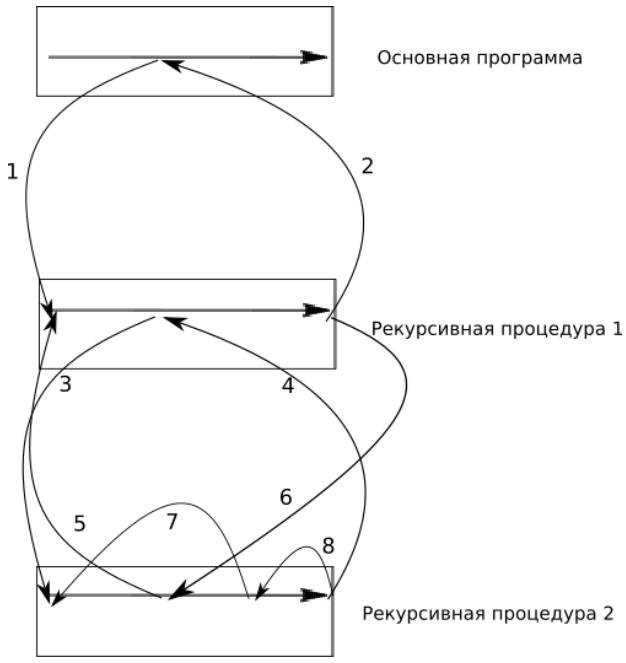

Рисунок 3. *Смешанная рекурсия (составлено авторами)*

Рассмотрим структуру реальной программы, в которой реализована схема из рисунка 3. Программа представлена в листинге 4. Это не что иное, как реализация алгоритма пузырьковой сортировки. Конечно, мы привыкли видеть алгоритм пузырьковой сортировки в виде двух вложенных циклов [2, 3, 10]. Но, как было показано выше, рекурсия представляет собой

обобщение циклического алгоритма. Вызов функции rec1 из rec2 это переход к следующему значению параметра внешнего цикла. Вызов же функции rec2 самой себя – переход к следующему значению параметра внутреннего цикла. Конечно, имитацию возврата из внутреннего цикла можно было бы реализовать и простым возвратом из функции rec2. Но мы ставили перед собой задачу реализовать пример косвенной рекурсии, с помощью которой может быть очень красиво сделана сортировка массива.

В представленной в листинге 4 программе есть три важных элемента, о которых мы говорили ранее: параметры, локальные переменные, глобальные переменные. Очень часто при реализации рекурсивных алгоритмов есть смысл уменьшить число передаваемых параметров и избавиться от использования локальных переменных (в нашем случае это локальная переменная t). Такая необходимость может возникнуть при сортировке очень больших массивов. И локальные переменные и параметры могут быть заменены глобальными переменными. Последнее может несколько усложнить алгоритм программы, но в результате мы можем получить значительную экономию стековой памяти.

Листинг 4. Реализация пузырьковой сортировки на основе смешанной рекурсии.

```
#include <stdio.h>
void rec1(int);
void rec2(int,int);
int a[100];
int n;
int main(){
	int k;
//количество элементов массива
	if(scanf("%d",&n)==EOF){
		return 1;
	}
//ввод элементов массива
	for(k=0; k<n; k++){
		if(scanf("%d",&a[k])==EOF){
			return 1;
		}
	}
//вызов рекурсивной функции
	rec1(0);
//вывод отсортированного массива
	for(k=0; k<n; k++){
		printf("%d ",a[k]);
	}
	return 0;
```


45PDMN418
Издательство «Мир науки» \ Publishing company «World of science» http://izd-mn.com

```
};
//функция создания очередного "пузырька"
void rec1(int m){
        if(m==n-1)return;
        rec2(m,0);
}
//функция всплытия очередного "пузырька"
void rec2(int l, int j){
        int t;
        if(j==n-l-1){
//переход к следующему "пузырьку"
                rec1(l+1);
        }else{
//всплытие "пузырька"
                if(a[j]>a[j+1]){
                        t=a[j]; a[j]=a[j+1]; a[j+1]=t;
                }
                rec2(l,j+1);
        }
        return;
}
```

Рекурсивные задачи

Важный вопрос, который неизбежно должен встать перед обучаемым – а какие типы задач предпочтительнее решать рекурсивно? Конечно, на первый взгляд, к задачам, которые удобнее всего решать рекурсивно, следует отнести задачи, в которых какие-либо значения, вычисляются на основе предыдущих. Типичным примером такой задачи, является вычисление чисел Фибоначчи. Последовательность чисел Фибоначчи (F_n) задается следующим соотношением: $F_0=0$, $F_1=1$, $F_n=F_{n-1}+F_{n-2}$, где n=>2. Действительно вычисление таких чисел очень хорошо ложится на рекурсивную технологию. Однако, заметим, что число Фибоначчи вычисляется на основе двух предыдущих чисел. Следовательно, легко построить алгоритм, чисто циклический, введя просто две переменные для хранения двух предыдущих значений данного элемента [2].

Наиболее эффективно рекурсивные алгоритмы работают в тех случаях, в которых требуется обходить сложные структуры, графически представляемые в виде сетевых или иерархических графов (иерархический граф является частным случаем сетевого графа). Типичным примером такой задачи является поиск наиболее оптимального маршрута, при наличии большого количества пересекающихся путей. При чем речь идет не о решении в конкретной ситуации, а об общем решении, при произвольном расположении дорог. В

идеальном случае речь может идти о прокладывании маршрута в лабиринте. Именно так формулируются типовые задания на олимпиадах по программированию.

Многие задачи, которые на первый взгляд не связаны с графами, в действительности можно описать с помощью графов. К такой задаче относится так называемая быстрая сортировка [10]. В словесной упрощенной форме алгоритм быстрой сортировки для одномерного числового массива можно описать следующим образом:

1. Выбирается один элемент массива, называемый опорным. Выбор опорного элемента не влияет на результат сортировки, но может повлиять на время сортировки.
2. Все элементы массива распределяются по отношению к опорному массиву следующим образом: элементы меньшие опорного располагаются слева (т. е. с меньшими индексами), элементы не меньшие опорного располагаются справа от него.
3. В результате массив оказывается разделенным на две части. Теперь для каждой части повторяем алгоритм, начиная с шага 1.

Легко видеть, что графически весь алгоритм может быть представлен в виде дерева, в узлах которого располагаются опорные элементы. Обход дерева и будет, в сущности, являться процессом сортировки. Рекурсия для такого алгоритма естественна и может быть реализована следующей программой (см. листинг 5).

Листинг 5. Пример реализации быстрой сортировки.

```
//быстрая сортировка
#include <stdio.h>
void rec1(int, int, int*);
void ins(int *, int, int);
int a[]={7,11,10,0,0,2,1,5,6,12,7,7};
int main(){
	int i,n=12;
//вызов рекурсивной функции
	rec1(0,n,a);
	for(i=0; i<n; i++)printf("%d ",a[i]);
	printf("\n");
	return 0;
};
//рекурсивная функция сортировки
void rec1(int b, int n, int *a){
	int i;
	if(n-b<=1)goto ex;
	i=b+1;
	while(i<n){
```

```
            if(a[i]<=a[b]){
                ins(a,i,b);
                b++;
            }
        i++;
        }
        rec1(0,b,a); rec1(b+1,n,a);
ex:
        return;
}
//функция пополнения группы элементов, меньших опорного
void ins(int * a, int m, int f){
        int j, e=a[m];
        for(j=m; j>f; j--){
            a[j]=a[j-1];
        }
        a[f]=e;
}
```

Сделаем пояснения к программе из листинга 5.

1. Рекурсивная функция rec1 имеет три параметра: начальный индекс первого элемента массива, количество элементов в массиве, указатель на массив. Массив по понятным причинам в функцию не передается (нам нужен исходный массив, а не его копия), передается лишь его адрес. Заметим, что использование параметров позволяет нам выполнять процесс сортировки над любым фрагментом массива, в том числе и путем ее рекурсивного вызова.
2. В качестве опорного элемента выбирается элемент, соседствующий с первым элементом фрагмента массива (i=b+1).
3. Функция вставляет элемент меньший опорного слева от него. При этом, поскольку часть элементов массива оказывается сдвинутой вправо на единицу, индекс опорного элемента также увеличивается на 1 (см. оператор b++).
4. Условием выхода из рекурсии является выполнение неравенства: n-b<=1. Использование оператора goto в таких ситуациях скорее оправдано, так как упрощает код.

Выводы

Сделаем несколько заключительных замечаний, касающихся темы рекурсивного программирования в школьных курсах Информатики и курсах Программирования в учреждениях высшего образования.

1. Тема «Рекурсивное программирование» должна быть выделена в общем курсе программирования.
2. Рекурсивное программирование следует рассматривать как технологию, обобщающую обычные циклические алгоритмы.
3. Введение понятие рекурсии легче всего осуществить рассматривая простые переборные задачи, сравнивая рекурсивный алгоритм с алгоритмом, на базе оператора цикла.
4. После сравнительного анализа цикла и рекурсии следует обратить внимание на тот факт, что рекурсии можно рассматривать как обобщение цикла и показать, где это обобщение проявляется.
5. Рассматривая простую и косвенную рекурсии, необходимо обратить внимание, что косвенная рекурсия является аналогом вложенных циклических конструкций.
6. Важным шагом в изучении рекурсивных алгоритмов является рассмотрение роли глобальных переменных, локальных переменных и параметров, указав на роль стековой памяти и возможности ее экономии.
7. Следует указать, что рекурсия позволяет решать сложные задачи, связанные с определением оптимальных путей в сетевых и иерархических структурах, решить которые с помощью циклических алгоритмов довольно сложно.

ЛИТЕРАТУРА

1. Лебедева, Т.Н. Формирование алгоритмического мышления школьников в процессе обучения рекурсивным алгоритмам в профильных классах средней общеобразовательной школы: дисс. на соискание ученой степени канд. пед. наук (13.00.02 теория и методика обучения информатике) / Т.Н. Лебедева. – Екатеринбург, 2005. – 219 с.
2. Вирт, Никлаус. Алгоритмы и структуры данных [Текст] / Никлаус Вирт. – М.: ДМК, 2010. – 274 с.
3. Ахо А., Хопкрофт Дж., Ульман Дж, Структуры данных и алгоритмы [Текст] / А. Ахо, Дж. Хопкрофт, Дж. Ульман, – М., Вильямс, 2007. – 400 с.
4. Дейкстра, Э. Дисциплина программирования [Текст] / Э. Дейкстра. – М.: Мир, 1978. – 275 с.
5. Вирт, Никлаус. Систематическое программирование [Текст] / Никлаус Вирт. – М.: Мир, 1977. – 183 с.
6. Пирогов В.Ю. Ассемблер и дизассемблирование. [Текст] / В.Ю. Пирогов – С.-П.: БХВ-Петербург, 2006. – 464 с.
7. Носов В.А. Основы теории алгоритмов и анализа их сложности [Текст] / В.А. Носов. – М., 1992 – 138 с.
8. Роджерс Х. Теория рекурсивных функций и эффективная вычислимость [Текст] / Х Роджерс. – М.: Мир, 1972. – 624 с.
9. Головешкин, В.А., Ульянов, М.В. Теория рекурсии для программистов [Текст] / В.А Головощекин, М.В. Ульянов, – М.: Физматлит, 2006. – 296 с.
10. Кнут, Дональд Ж. Искусство программирования [Текст]. Т. 1. Основные алгоритмы / Дональд Ж. Кнут. – М.: Вильмс, 2015. – 720 с.

Pirogov Vladislav Jurievich
Shadrinsk pedagogical university, Shadrinsk, Russia
E-mail: Vladislav-133@yandex.ru

Balandina Irina Victorovna
Shadrinsk pedagogical university, Shadrinsk, Russia
E-mail: piv_vip@mail.ru

Foundations of teaching methods of recursive programming

Abstract. The paper discusses some issues of programming training. In particular it is examined elements of teaching methods of recursive programming. The authors propose to highlight the recursive programming as an independent topic and to consider it along with cyclic algorithms and procedural programming. The paper deals with in detail types of recursion and typical program structures. Authors, along with simple and indirect recursion, also distinguish a mixed recursion. It is shown that the usual cyclic algorithm can be represented in a recursive form. The authors propose to consider the recursive algorithm as a generalization of the cyclic algorithm. Further the paper gives examples of typical recursive algorithms, in particular, the problem of quick sorting is considered. Author's provisions are illustrated by the program code in the algorithmic language C with detailed comments. At the end of the paper some general recommendations for teaching recursive programming are given:

1. The theme “Recursive programming” should be highlighted in the general course of programming training.
2. The recursive programming should be considered as a technology that generalizes conventional cyclic algorithms.
3. The concept of recursion is easiest to introduce, considering simple problems solved by cyclic search.
4. Indirect recursion in particular can be considered as a generalization of nested loops.
5. It is important to consider memory savings and the role of global and local variables, as well as parameters.
6. The recursive approach makes it easy to solve complex problems that are difficult to solve in other ways.

Keywords: programming; teaching methods of computer science; recursion; computer science; recursive programming; procedural programming; quicksort