Лабораторная работа № 14: РЕШЕНИЕ ЗАДАЧ С ПРИМЕНЕНИЕМ РЕКУРСИВНЫХ АЛГОРИТМОВ И ПО ПОИСКУ ЭЛЕМЕНТОВ И СОРТИРОВКАМ В ДВУМЕРНЫХ МАССИВАХ

(4 часа)

Цель лабораторной работы – освоить основные принципы по построению рекурсивных алгоритмов, научиться применять методы сортировок к двумерным массивам, методы слияния отсортированных массивов.

План лабораторной работы

1. Изучить понятия рекурсии, рекурсивные функции в программировании, приемы построения рекурсивной триады при решении задач.
2. Научиться применять рекурсивные методы в решении задач.
3. Закрепит навыки применения рекурсивных алгоритмов сортировок массива.
4. Изучить особенности применения алгоритмов сортировок и перестановок в двумерных массивах.

1. Необходимые теоретические сведения к выполнению лабораторной работы.

Рекурсия — это такой способ организации обработки данных, при котором программа вызывает сама себя непосредственно, либо с помощью других программ.

Итерация — способ организации обработки данных, при котором определенные действия повторяются многократно, не приводя при этом к рекурсивным вызовам программ.

Математическая модель рекурсии заключается в вычислении рекурсивно определенной функции на множестве программных переменных. Примерами таких функций могут служить факториал числа и числа Фибоначчи. В каждом из этих случаев значение функции для всех значений аргумента, начиная с некоторого, определяется через предыдущие значения.

Любой алгоритм, реализованный в рекурсивной форме, может быть переписан в итерационном виде, и наоборот. Данное утверждение не означает, что временная и емкостная эффективности получающихся программ обязаны совпадать. Однако наилучшие рекурсивный и итерационный алгоритм имеют совпадающую с точностью до постоянного множителя временную сложность.

Пеример* 1*: В арифметической прогрессии найдите a_n, если известны a_1 = -2.5, d =0.4, не используя формулу n -го члена прогрессии.

Формализация алгоритма в математической постановке

По определению арифметической прогрессии, $a_n=a_n-1+d$, при этом:

Таким образом, нахождение a_n для номера n сводится к решению аналогичной задачи, но только для номера *n -1*, что в свою *очередь* сводится к решению для номера *n -2*, и так далее, пока не будет достигнут номер 1 (*значение* a_1 дано *по* условию задачи).

Код рекурсивной функции имеет следующий вид:

```
float arifm (int n, float a, float d)
 {
 if (n<1) return 0;         // для неположительных номеров
 if (n==1) return a;        // базовый случай: n=1
 return arifm(n-1, a ,d) + d; // общий случай
 }
```

Для решения задач рекурсивными методами разрабатывают следующие этапы, образующие *рекурсивную триаду*:

- *параметризация* – выделяют параметры, которые используются для описания условия задачи, а затем в решении;
- *база рекурсии* – определяют тривиальный случай, при котором решение очевидно, то есть не требуется обращение функции к себе;
- *декомпозиция* – выражают общий случай через более простые подзадачи с измененными параметрами.

Целесообразность применения рекурсии в программировании обусловлена спецификой задач, в постановке которых явно или опосредовано указывается на возможность сведения задачи к подзадачам, аналогичным самой задаче. При этом эффективность рекурсивного или итерационного способов решения одной и той же задачи определяется в ходе анализа работоспособности программы на различных наборах данных. Таким образом, рекурсия не является универсальным способом в программировании. Ее следует рассматривать как альтернативный вариант при разработке алгоритмов решения задач.

Повысить эффективность рекурсивных алгоритмов часто представляется возможным за счет пересмотра этапов триады. Например, введение дополнительных параметров, не оговоренных в условии задачи, в реализации декомпозиции могут быть применены другие соотношения, а также можно организовать расширение базовых случаев с сохранением промежуточных результатов.

Рекурсивный стек - это область памяти, предназначенная для хранения всех промежуточных значений локальных переменных, при каждом следующем рекурсивном обращении. Для каждого текущего обращения

формируется локальный слой данных стека (при этом совпадающие идентификаторы разных слоев стека независимы друг от друга и не отождествляются).

Завершение вычислений происходит посредством восстановления значений данных каждого слоя в порядке, обратном рекурсивным обращениям. В силу подобной организации количество рекурсивных обращений ограничено размером области памяти, выделяемой под программный код. При заполнении всей предоставленной области памяти попытка вызова следующего рекурсивного обращения приводит к ошибке переполнения стека.

Сформулируем *ключевые термины* по рекурсивным алгоритмам

База рекурсии – это тривиальный случай, при котором решение задачи очевидно, то есть не требуется обращение функции к себе.

Декомпозиция – это выражение общего случая через более простые подзадачи с измененными параметрами.

Косвенная (*взаимная*) рекурсия – это последовательность взаимных вызовов нескольких функций, организованная в виде циклического замыкания на тело первоначальной функции, но с иным набором параметров.

Параметризация – это выделение из постановки задачи параметров, которые используются для описания условия задачи и решения.

Прямая рекурсия – это непосредственное обращение рекурсивной функции к себе, но с иным набором входных данных.

Рекурсивная триада – это этапы решения задач рекурсивным методом.

Рекурсивная функция – это функция, которая в своем теле содержит обращение к самой себе с измененным набором параметров.

Рекурсивный алгоритм – это алгоритм, в определении которого содержится прямой или косвенный вызов этого же алгоритма.

Рекурсивный стек – это область памяти, предназначенная для хранения всех промежуточных значений локальных переменных при каждом следующем рекурсивном обращении.

Рекурсия в программировании – это пошаговое разбиение задачи на подзадачи, подобные исходной.

Рекурсия в широком смысле – это определение объекта посредством ссылки на себя.

Пример 2. Для целого неотрицательного числа *n* найдите его *факториал*.

Разработаем рекурсивную триаду.

Параметризация: n – неотрицательное целое число.

База рекурсии: для n =0 факториал равен 1.

Декомпозиция: n!=(n-1)! × n.

Коды соответствующих рекурсивной `factor_r` и итерационной `factor_i` функций представлены ниже:

```
long factor_r(int n)
{
	if (n < 0) return 0; // для отрицательных чисел
	if (n == 0) return 1; // базовый случай: n=0
	return factor_r(n - 1)*n; // общий случай (декомпозиция)
}
long factor_i(int n)
{
	int p=1;
	for (int i = 1; i <= n; i++)
		p *= i;
	return p;
}
```

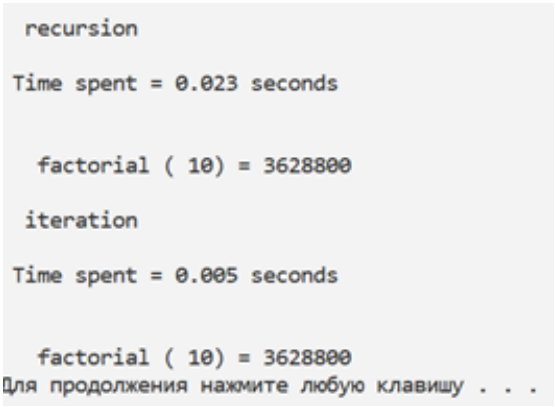
Оценим скорость выполнения итерационного и рекурсивного алгоритмов вычисления факториала. Для этого создадим функцию, которая будет многократно обращаться сама к себе, чтобы искусственно накопить время выполнения каждой функции и увидеть разницу этого времени, если она будет.

```
void time_estimation(long(*pf)(int), int N_iter, int N)
{
	int mf;
	clock_t begin = clock();
	for (int i = 1; i < N_iter; i++)
		mf = pf(N);
	clock_t end = clock();// число тиков перед началом работы
фрагмента программы
	double time_spent = (double)(end - begin) /
CLOCKS_PER_SEC;//время на расчёт в мс
	cout << "\n Time spent = " << time_spent << " seconds\n" << endl;
	cout << "\n   factorial ( "<<N<<") = " << mf << endl;
}
```

Соответствующая главная программа, реализующая поставленную задачу, имеет вид:

```
int main()
{
int N_iter = 100000, N=10;
cout << "\n  recursion\n " ;
time_estimation(factor_r, N_iter, N);
cout << "\n  iteration \n ";
time_estimation(factor_i, N_iter, N);
system("pause");
return 0;
}
```

Результаты работы программы, представленные следующим скриншотом:



показывают, что в данном случае итерационный алгоритм работает примерно в пять раз быстрее, чем рекурсивный. Однако существуют задачи, где рекурсивные алгоритмы более эффективны, чем итерационные. Основное достоинство рекурсивных алгоритмов в том, что во многих случаях они дают гораздо более элегантное решение по сравнению с итеративным методом, распространенным примером является обход двоичного дерева – задача, которая будет рассмотрена в специализированных курсах по алгоритмам. В общем, итеративные версии обычно немного быстрее (и во время оптимизации вполне могут заменить рекурсивную версию), но рекурсивные версии проще понять и правильно реализовать.

Таким образом, достоинством рекурсии является компактная запись, а недостатками — расход времени и памяти на повторные вызовы функции и передачу ей копий параметров, и, главное, опасность переполнения стека.

Рекурсивные методы лежат в основе многих типов сортировок массивов, относящихся к разряду *быстрых сортировок*. Рассмотрим один из примеров такой сортировки в задаче с двумерными массивами.

Пример 3. Создать двумерный целочисленный массив размера n×m, где n и m вводятся с консоли. Значения элементов массива определяются как цифры с позиции 3 до позиции 6 синуса от выражения $sin(i+j)+cos(i \times j)$, где i и j - индексы текущего элемента массива. Используя алгоритм рекурсивной сортировки, отсортировать его столбцы по возрастанию. Исходный и отсортированный массивы вывести на консоль в виде пропорциональной таблицы.

Решение. В силу особенности представления двумерных массивов в виде таблиц смысл сортировки такого массива сводится к упорядочиванию элементов, объединенных в столбцы или строки. Например, сортировка по возрастанию элементов столбцов означает, что элементы следует расположить по возрастанию сверху вниз в каждом столбце отдельно. При этом,

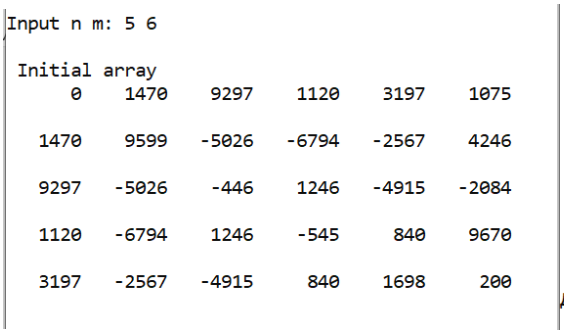
рассматривая строку или столбец как одномерный массив, к ним применяют алгоритмы сортировок одномерных массивов. Решение задачи основано на применении одного из методов быстрой сортировки, основанного на рекурсивной функции сортировки (функции - `merg_series`, `merg_rec`, `merge_sort`). В приведённом примере следует обратить внимание на функцию `init_array`, которая возвращает указатель на указатель при инициализации массива.

```
void merg_series(int *A, int b, int c, int e, int *D)//  слияние серий
void merg_rec(int *A, int b, int e, int *D);//рекурсивный алгоритм сортировки слиянием
void merge_sort(int *A, int n);//функция-обёртка для рекурсивной функции
int** init_array(int n, int m);// инициализация двумерного массива
void print2(int **a, int n, int m); // вывод на консоль двумерного массива
void sort_array2(int **B, int n, int m);// Сортировка столбцов массива

int main()
{
	int **A, n, m;
	cout << "Input n m: "; cin >> n >> m;
	A = init_array(n, m);
	cout << "\n Initial array\n";
	print2(A, n, m);
	sort_array2(A, n, m);
	cout << "\n Sorted array\n";
	print2(A, n, m);
	system("pause");
	return 0;
}
void merg_series(int *A, int b, int c, int e, int *D)
{
	int i = b, j = c + 1, k;
	for (k = b; k <= e; k++)
		if (j > e) D[k] = A[i++];
		else if (i > c) D[k] = A[j++];
		else if (A[i] <= A[j]) D[k] = A[i++];
		else D[k] = A[j++];
	return;
}
void merg_rec(int *A, int b, int e, int *D)
{
	int c = (b + e) / 2;
	if (b < c) merg_rec(A, b, c, D);
	if (c + 1 < e) merg_rec(A, c + 1, e, D);
	merg_series(A, b, c, e, D);
	for (int i = b; i <= e; i++)
		A[i] = D[i];
	return;
}
```

```
void merge_sort(int *A, int n)
{
	int *D = new int[n];
	merg_rec(A, 0, n - 1, D);
	delete[] D;
}
int** init_array(int n, int m)
{
	int **B = new  int*[n];
	for (int j = 0; j < n; j++)
		B[j] = new int[m];
	for (int i = 0; i < n; i++)
		for (int j = 0; j < m; j++)
		{
			B[i][j] = (sin(i + j) + cos(i*j))*1e6;
			B[i][j] %= 10000;
		}
	return  B;
}
void print2(int **a, int n, int m)
{
	for (int i = 0; i < n; i++)
	{
		for (int j = 0; j < m; j++)
			printf("%7d ", a[i][j]);
			cout << endl << endl;
	}
	return;
}
void sort_array2(int **B, int n, int m)
{
	int *a = new int[n];
	for (int i = 0; i < m; i++)
	{
		for (int j = 0; j < n; j++)
			a[j] = B[j][i];
		merge_sort(a, n);
		for (int j = 0; j < n; j++)
			B[j][i] = a[j];
	}
	delete[] a;
	return;
}
```

Решение имеет следующий вид.



```
Input n m: 5 6

Initial array
     0    1470    9297    1120    3197    1075

  1470    9599   -5026   -6794   -2567    4246

  9297   -5026    -446    1246   -4915   -2084

  1120   -6794    1246    -545     840    9670

  3197   -2567   -4915     840    1698     200
```

```
Sorted array
     0   -6794   -5026   -6794   -4915   -2084

  1120   -5026   -4915    -545   -2567     200

  1470   -2567    -446     840     840    1075

  3197    1470    1246    1120    1698    4246

  9297    9599    9297    1246    3197    9670

Для продолжения нажмите любую клавишу . . .
```

Анализ решения показывает, что в программе сделаны необходимые сортировки столбцов и таблицы представлены в пропорциональном виде, благодаря применению форматированного вывода на консоль.

Краткие итоги

– Свойством рекурсивности характеризуются объекты окружающего мира, обладающие *самоподобием*.

– Рекурсия в широком смысле характеризуется определением объекта посредством *ссылки на себя*.

– *Рекурсивные функции* содержат в своем теле обращение к самим себе с измененным набором параметров. При этом обращение к себе может быть организовано через цепочку взаимных обращений функций.

– Решение задач рекурсивными способами проводится посредством разработки *рекурсивной триады*.

– *Целесообразность* применения рекурсии в программировании обусловлена спецификой задач, в постановке которых явно или опосредовано указывается на возможность сведения задачи к подзадачам, аналогичным самой задаче.

– Область памяти, предназначенная для хранения всех промежуточных значений *локальных переменных*, при каждом следующем рекурсивном обращении, образует *рекурсивный стек*.

– *Рекурсивные методы* решения задач нашли широкое применение в процедурном программировании.

– *Рекурсивные методы* позволяют реализовать *быстрые* сортировки в массивах.

2. Методика выполнения самостоятельной работы

1. ***Внимательно изучить примеры***, приведенные в методической части лабораторной работы. Изучить методы анализа результатов и их описание. Особое внимание уделить разработке *рекурсивной триады*.

2. Ознакомиться с условием задачи и примерами решений аналогичных задач из первой части лабораторной работы.
3. Составить контрольный пример.
4. Проверить полноту задачи: рассмотреть все возможные исходы решения в зависимости от исходных данных, предусмотреть случаи возможного зависания, зацикливания программы и запрограммировать корректную реакцию программы на эти ситуации.
5. Записать словесный алгоритм или составить блок-схему алгоритма.
6. Записать код программы на C^{++}.
7. Запустить программу, провести синтаксическую отладку.
8. Проверить работоспособность программы путём сравнения результатов с контрольным примером на все возможные случаи исходных данных.
9. Завершить работу составлением Отчёта, где будут описаны все этапы выполнения самостоятельного задания и приведены распечатки консольного вывода. Образец отчета приведен в Приложении.

3. Задания для самостоятельной работы

Номер варианта выбирается по формуле N%6+1, где N - порядковый номер студента в списке группы.

Задание 1. Выполните приведенные ниже задания по соответствующему варианту. В Отчёте по лабораторной работе *обязательно* привести контрольные примеры *и построенную рекурсивную тираду*.

Некоторые варианты Задания 1 даны в двух типах – простой (а) и посложнее (б) – выбирайте на своё усмотрение.

Вариант 1а. Определите закономерность формирования членов последовательности. Найдите *n* -ый член последовательности: 1, 1, 2, 3, 5, 8, 13, ...

Вариант 1б. Имеется некоторая сумма денег *S* и набор монет с номиналами a_1, …, a_n. Монета каждого номинала имеется в единственном экземпляре. Необходимо найти все возможные способы разменять сумму *S* при помощи этих монет.

Вариант 2. Составьте программу вычисления биномиального коэффициента C_m^n для данных неотрицательных целых *m*, *n* ($m \geq n$): $C_m^n = \frac{m!}{n! \cdot (m-n)!}$. Решите задачу двумя способами: 1 – используйте функцию вычисления факториала; 2 – выразите вычисление C_m^n через C_{m-1}^{n-1}. Сравните эффективность по времени выполнения для обоих типов алгоритмов.

Вариант 3. Исполнитель умеет выполнять два действия: "+1", "*2". Составьте программу получения из числа 1 числа 100 за наименьшее количество операций.

Вариант 4а. Найдите наибольший общий делитель (НОД) двух натуральных чисел с помощью алгоритма Евклида.

Указание: пример алгоритма нахождения НОД делением.

1. Большее число делим на меньшее.
2. Если делится без остатка, то меньшее число и есть НОД (следует выйти из цикла).
3. Если есть остаток, то большее число заменяем на остаток от деления.
4. Переходим к пункту 1

Вариант 4б. Дано натуральное число *N*. Выведите все его цифры по одной, в обратном порядке, разделяя их пробелами или новыми строками. При решении этой задачи нельзя использовать строки, списки, массивы, циклы. Разрешена только рекурсия и целочисленная арифметика.

Вариант 5а. Разработайте программу вычисления a^n натуральной степени *n* вещественного числа *a* за наименьшее число операций.

Вариант 5б. Разработайте рекурсивную функцию для получения суммы цифр натурального числа N. При решении этой задачи нельзя использовать строки, списки, массивы, циклы. Разрешена только рекурсия и целочисленная арифметика.

Вариант 6. Дано натуральное число *N*>1. Составить рекурсивную программу-функцию проверяющую, является ли заданное натуральное число *N* простым. Указание: Задача сама по себе не рекурсивна, т.к. проверка числа N на простоту никак не сводится к проверке на простоту меньших чисел. Поэтому нужно сделать еще один параметр рекурсии: делитель числа, и именно по этому параметру и делать рекурсию: *recursion* (int *N*, int *i*), где *i*-дополнительный параметр. При вызове должен быть равен 2.

Задание 2. Создать двумерный массив A_{nm}, где $n \in [4;7]$ и $m \in [5;8]$ - случайные числа в заданных диапазонов. Инициализировать массив по правилу соответствующего варианта, описанному в Лабораторной работе 13 для целочисленного массива A.

Вывести на консоль полученный массив.

Выполнить операции с созданным массивом в соответствии со своим вариантом. Во всех функциях исходный массив будет изменяться. Для сортировки использовать рекурсивную функцию.

Вариант 1. Разработать функцию, которая сортирует каждую строку массива по убыванию. Распечатать преобразованный массив.

Вариант 2. Разработать функцию, которая переставит столбцы массива так, чтобы их максимальные элементы образовали возрастающую последовательность. Вывести массив после перестановки.

Вариант 3. Разработать функцию, которая отсортирует весь массива по убыванию, т.е. так, чтобы $A_{00} \geq A_{01} \geq \ldots \geq A_{nm}$. Вывести массив после сортировки.

Вариант 4. Разработать функцию, которая отсортирует каждый столбец массива по убыванию. Затем переставить столбцы массива в порядке возрастания первых элементов каждого столбца. Вывести массив после сортировки и после перестановки столбцов.

Вариант 5. Разработать функцию, которая отсортирует строки массива по невозрастанию и расставит их в порядке возрастания минимальных элементов. Вывести массив после сортировки и после перестановки строк.

Вариант 6. Разработать а) функцию, которая превратит массив в квадратную матрицу, б) функцию, которая реализует обмен значениями элементов диагоналей квадратной матрицы, расположенных в одной строк, с) отсортирует элементы строк массива, расположенные в верхнем треугольнике между диагоналями, по возрастанию. Выводить квадратный массив после каждой манипуляции с его элементами.

* * *

4. Контрольные вопросы

1. Почему при правильной организации рекурсивные вызовы не зацикливаются?
2. Почему не отождествляются совпадающие идентификаторы при многократных рекурсивных вызовах?
3. Почему рекурсивные обращения завершаются в порядке, обратном вызовам этих обращений?
4. Чем ограничено при выполнении программы количество рекурсивных вызовов?
5. Что такое рекурсивная триада?
6. Какие рекурсивные методы сортировок вы знаете?
7. В чем принципиальное отличие задач сортировок двумерных и одномерных массивов?

5. Домашнее задание

1. Дано натуральное число, не выходящее за пределы типа unsigned int. Число представлено в десятичной системе счисления. Разработайте рекурсивную функцию для перевода его в систему счисления с основанием 16.

Указание: Пусть требуется перевести целое число n из десятичной в p-ичную систему счисления (по условию задачи, $p = 16$), то есть найти такое k, чтобы выполнялось равенство $n_{10} = k_p$.

Параметризация: n – данное натуральное число, p – основание системы счисления.

База рекурсии: на основании правил перевода чисел из десятичной системы в систему счисления с основанием p, деление нацело на основание системы выполняется до тех пор, пока неполное частное не станет равным нулю, то есть: если целая часть частного n и p равна нулю, то $k = n$. Данное условие

можно реализовать иначе, сравнив n и p: целая часть частного равна нулю, если $n < p$.

Декомпозиция: в общем случае k формируется из цифр целой части частного n и p, представленной в системе счисления с основанием p, и остатка от деления n на p.

2. Две группы студентов прошли тестирование по 100 бальной системе по Математике, Программированию, Физике и Химии. Количество студентов в группах разное. Сформировать список из N лучших студентов обеих групп по каждому из предметов для участия в соответствующих предметных олимпиадах. Указание: использовать необходимые алгоритмы по работе с отсортированными массивами из Лекции 15.