Лабораторная работа № 13: РЕШЕНИЕ ЗАДАЧ ПО ПОИСКУ ЭЛЕМЕНТОВ И СОРТИРОВКАМ В МАССИВАХ. РАБОТА С ФАЙЛАМИ

(4 часа)

Цель лабораторной работы – освоить методы работы с файлами, записи и чтения из них информации, изучить основные способы поиска и сортировки в массивах, научиться оценивать их эффективность произвести оценку эффектность.

План лабораторной работы

1. Изучение способов записи массива в файл.
2. Познакомится с возможностью возвращения указателя на массив, как результат работы функции;
3. Закрепит навыки передачи функций как параметров.

1. Примеры решения задач по заявленной теме

Задача 1: Дана функция в виде ряда S(x, ε)=$x + \sum_{i=1}^{\infty}(-1)^n \frac{(i+1)!}{(2i+1)!(2i-1)} x^{2i+1}$, вычисляемого с заданной точностью ε для |x|<1. Построить две программы для расчёта ряда: одну - с применением формулы общего члена ряда, предусмотрев возможные ситуации переполнения, другую – с применением рекуррентной формулы. Построить массив из значений этой функции на интервале $x \in [x_n; x_k]$ с шагом Δx. Результаты записать в файл. Оценить время работы программы с применением двух типов функций. Найти и напечатать в массиве первый член, который совпадает по значению с точностью $\bar{\varepsilon}$ с последним элементом массива, если таковых нет, выдать об этом сообщение.

Разработка алгоритма.

Вначале разобьём алгоритм на несколько шагов, в соответствии с тем, на сколько отдельных модулей мы можем распределить решение всей задачи.

1. Очевидно, что вначале необходимо построить коды функций, реализующих расчёт функции S(x, ε) по двум разным алгоритмам: пусть S_c(x,eps) – использует формулу общего члена, а S_t(x,eps) – рекуррентные соотношения.
2. На следующем шаге необходимо задать параметры определения аргумента x - $x_n; x_k, \Delta x$ и вычислить значение функции на каждом шаге заданного интервала.
3. При выполнении шага 3 необходимо рассмотреть возможность занесения данных для хранения в массив A[na]. Для этого необходимо вычислить размер na будущего массива, по заданным параметрам

аргумента x: $na = \left[\frac{xk\ xn}{\Delta x}\right] + 1$ и выделить соответствующую динамическую память. Функция, которая будет инициализировать массив, назовём Init_Arr и она будет зависеть от параметров xn, xk, dx.

4. Функция Init_Arr должна не только инициализировать массив, но и определить его размер и передать вызывающей программе указатель на массив. Поэтому мы в качестве возвращаемого значения и передадим этот указатель.
5. Функция Init_Arr реализует инициализацию массива через две функции S_c и S_t, поэтому надо предусмотреть в ней параметры для принятия имени функции и значения eps.
6. Поскольку в задаче определена цель определить эффективность обеих функций S_c и S_t в код функцию Init_Arr необходимо заложить возможность оценки времени инициализации массива по разным функциям.
7. Результаты работы программы Init_Arr необходимо сохранить в файле, возможные размеры массивов могут быть достаточно большими.
8. После записи массива в файл можно приступить к поиску элемента, наиболее близкого к последнему элементу A[na-1] с точностью $\bar{\varepsilon}$.

Рассмотрим коды программ, реализующие решение шагов 1-7.

```
#include<iostream>
#include<ctime>
#include<fstream>

using namespace std;
double S_t(double x, double eps); // расчёт заданного ряда по
рекуррентным формулам
double S_c(double x, double eps); // расчёт заданного ряда по формуле
общего члена
double fact(int k);// вычисление факториала
double *Init_Arr(int &na, double xn, double xk, double dx,
double(*pf)(double, double), double eps);// функция инициализации
массива с помощью функции, передаваемой по указателю (*pf), аргумент
задается в заданном интервале с заданным шагом.
void Out_f_Arr(double *A, int &na); //вывод массива в файл

int main()
{
      double *A;
      double xn, xk, dx, eps;
      int na;
      cout << "Input parameters for argument xn xk dx:"; cin >>xn >> xk
>> dx;
      cout << "Input eps:"; cin >>eps;
      cout << "\n common term used \n";

      A = Init_Arr(na, xn, xk, dx, S_c,eps);
```

```
	Out_f_Arr(A, na);

	cout << "\n recurrence formula used\n";
	A = Init_Arr(na, xn, xk, dx, S_t, eps);

	Out_f_Arr(A, na);
	system("pause");
	return 0;
}
double S_t(double x, double eps)
{
	if (abs(x) >= 1) {
		cout << "\n S_t: |x|>=1"; system("pause"); exit(1);
	}
	double  ch = 1, zn = 1, u = x,  x2 = x * x, s = x, U;
	int i = 1;
	do {ch *= (i+1);
		zn *= (2 * i + 1)*(2 * i);
		u*= -x2;
		U = u * ch / zn / (2 * i - 1);
		s += U;
		i++;
	} while (abs(U) > eps);

	return s;
}
double S_c(double x, double eps)
{
	if(abs(x)>=1) {
		cout << "\n S_c: |x|>=1"; system("pause"); exit(1);
	}
	double u,  s = x;
	int i = 1;
	do {
		u = pow(-1, i)*pow(x, 2 * i + 1)*fact(i + 1) / fact(2 *
i+1)/ (2 * i - 1);
		s += u;
		i++;
		} while (abs(u) > eps);

	return s;
}
// для факториала используем стандартный алгоритм накопления
произведения, а не взятой из сети алгоритм рекурсивной функции.
   int fact(int k)
   {
	int p = 1;
	for (int i = 1; i <= k+0.5; i++)
		p *= i;
	if (p > 30000)
	{
```

```
            cout << "\n factorial - overflow!\n"; system("pause");
   exit(1);
      }
      return p;

double *Init_Arr( int &na, double xn, double xk, double dx, double
(*pf)(double, double), double eps)
{
      clock_t begin = clock();// число тиков в начале работы  программы
      na = (xk - xn) / dx + 1;
      double *A = new double[na];
      double x = xn;
      for (int i = 0; i < na; i++, x+=dx)
            A[i] = pf(x, eps);
      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;
      return A;
}
void Out_f_Arr(double *A, int na, bool flag)
{
      If (flag)
 {ofstream out("Lab7a.txt",ios::app); // создание и открытие файла для
записи
      if (!out)  // проверка успешности открытия файла
      {cout << "\n Нет файла  OUT\n";  system("pause"); exit(1);}

      for (int i = 0; i < na; i++)
            out << A[i] << ' ';
      out << endl<<endl<<endl;
      out.close();     // закрытие файла
}
Else
}
```

Анализ результатов.

1. Следуя логике алгебраического выражения для ряда S в случае применения целочисленных переменных при вычислении факториала, код программы которого приведён выше, даже при невысокой точности ε вычисления ряда происходит ситуация переполнения

```
Input parameters for argument xn xk dx:0.2 0.9 0.05
Input eps:0.001

 common term used

 factorial - overflow!
Для продолжения нажмите любую клавишу . . .
```

Поэтому приходится переходить к вещественным переменным, чтобы алгоритм работал для более широкого диапазона значений точности.

```
double fact(int k)
{
	double p = 1;
	for (double i = 1; i <= k+0.5; i++)
		p *= i;
	if (p > 1e35)
	{
		cout << "\n factorial - overflow!\n"; system("pause");
exit(1);
	}
	return p;
}
```

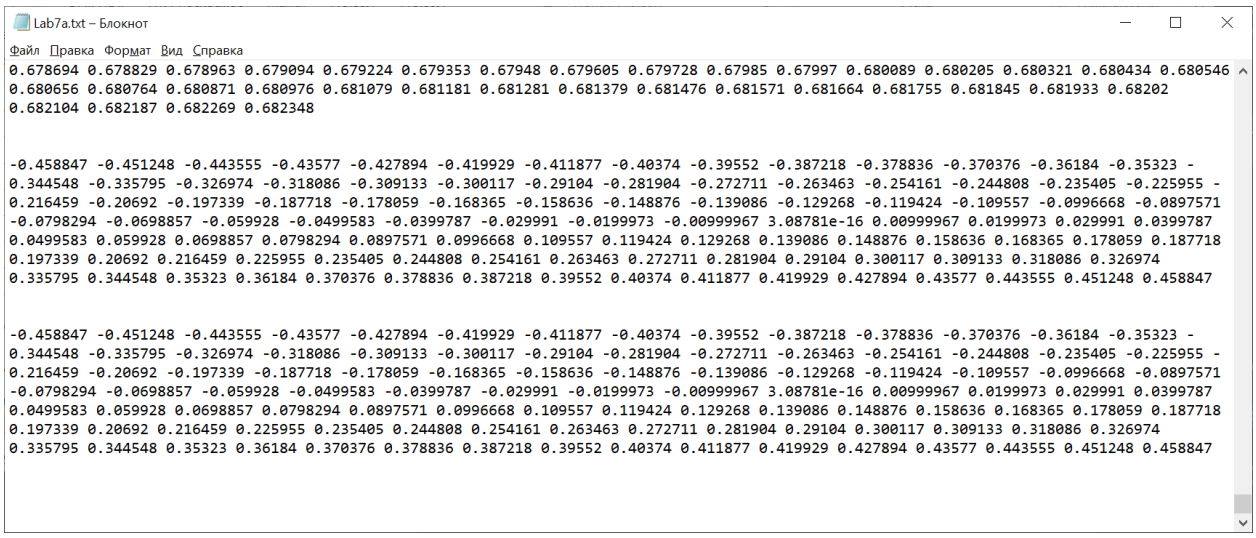
При переходе к типу **double** задача выполняется уже при достаточно высокой точности. Мы видим, что эффективность применения рекуррентных формул в 5 раза выше, чем применение метода расчёта по формуле общего члена ряда.

```
Input parameters for argument xn xk dx:-0.999 0.999 0.001
Input eps:1e-7

 common term used

 Time spent = 0.005 seconds


 recurrence formula used

 Time spent = 0.001 seconds

Для продолжения нажмите любую клавишу . . .
```

При заданных параметрах в файле накапливаются значения для 20 тысяч элементов массива. Чтобы посмотреть содержимое файла `"Lab7a.txt"` и оценить, существуют ли расхождения между расчётами массива по разным алгоритмам, необходимо уменьшить количество членов массиве, изменив или шаг, или границы интервала для аргумента *x*.

```
Input parameters for argument xn xk dx:-0.5 0.5 0.01
Input eps:1e-11
```

Данные решения при таких параметрах представлены ниже:



Два блока чисел представляют собой *одинаковые* значения массива, рассчитанные по двум разным функциям, но при большом объёме массива, время расчёта по разным функциям, как мы уже видели, разное.

Следует обратить внимание, что запись в файл, функции **Out_f_Arr** реализуется с применением модификатор `app`, что приводит к накоплению в файле предыдущих расчётов. Если этого не надо, то следует его удалить из оператора:

```
ofstream out("Lab7a.txt",ios::app);
```

Теперь приступим к решению 8-го шага словесного алгоритма. Для поиска наиболее близкого к последнему элементу массива применим два разных алгоритма поиска – метод простого перебора и метод дихотомии.

Дополним набор функций методами поиска

// Метод простого перебора

```
int find_double(double* A, int na, double p, double eps)
{
	for (int i = 0; i < na; i++)
		if (abs(A[i] - p) < eps) return i;
		return -1;
}
```

// Метод деления массива пополам - метод дихотомии

```
int bin_search(double* A, int na, double p, double eps)
{
	int ib = 0, ie = na - 1, ic;
	while (ib < ie)
	{
		ic = (ib + ie) / 2;
```

```
            if (A[ic] < p) ib = ic + 1;
            else ie = ic;
      }
            if (abs(A[ib] - p) < eps) return ib;
      return -1;
}
```

Сравнение методов по времени реализуем так же, как в случае сравнения скорости вычисления ряда разными методами. Составим функцию, в которую можно передавать разные методы поиска для обработки массива и оценивать соответствующее время выполнения поиска:

```
void search(double* A, int na, int(*pf)(double*, int, double, double),
double p, double eps)
{
      clock_t begin = clock(); // число тиков в начале работы
программы
      int ia = pf(A, na - 1, A[na - 1], eps);
            if (ia < 0)cout << "\n no terms equal to A[last]\n";
            else cout << "A[" << ia << "] = " << A[ia] << " ~  A[last]
=" << A[na - 1];
      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*1000 << "
milliseconds\n" << endl;

}
```

В этой программе реализованы две функции: оценка времени расчёта и вывод результатов поиска. Обратите внимание(!), что время выводится в миллисекундах. Фрагмента главной программы, обеспечивающей вызов функции, состоит из следующих операторов:

```
// searching for the nearest term to the last term of A
      cout << "\n Simple search \n";
      search(A, na-1, find_double, A[na-1], 1e-3);
      cout << "\n Dichotomous search \n";
      search(A, na - 1, bin_search, A[na - 1], 1e-3)
```

Как видим, точность совпадения $\bar{\varepsilon} = 0{,}001$ здесь задана невысокой. Поэтому разные методы могут дать и разные значения. Анализ сравнения результатов, представленных ниже, показывает, это, действительно, так и получилось: метод последовательного перебора нашёл число, которое имеет номер 1986, а метод дихотомии – 1996, которое оказалось ближе к последнему элементу массива. Выведенное время, по-видимому, отличается в долях миллисекунды, которые просто не могут быть выведены в силу ограниченности длины числа `time_spent`.

```
Input parameters for argument xn xk dx:-0.999 0.999 0.001
Input eps:1e-7

 common term used

 Time spent = 0.003 seconds


 recurrence formula used

 Time spent = 0.001 seconds


 Simple search
A[1986] = 0.681281 ~  A[last] =0.682269
 Time spent = 2 milliseconds


 Dichotomous search
A[1996] = 0.682187 ~  A[last] =0.682269
 Time spent = 2 milliseconds
```

2. Методика выполнения самостоятельной работы

1. ***Внимательно изучить пример***, приведенный в методической части лабораторной работы. Изучить методы анализа результатов и их описание.
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.

2. Задания для самостоятельной работы

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

Задание 1.

1. Создать с помощью генератора случайных чисел два массива A и B из $N_1 \in [50; 100]$ и $N_2 \in [1000; 2000]$ элементов, соответственно, и инициализируемых по заданному ниже правилу.
2. Вывести полученные массивы в разные файлы. В отчёт представить скрин начала файлов
3. Вывести на консоль каждый *m*-й элемент массива, при этом *m* выбирается для каждого массива так, чтобы не перегружать консоль, то есть так, чтобы элементы массива занимали на консоли не более 10 строк. Массивы выводить на консоль вместе с их индексами. Задачи вывода массива в файл и на консоль можно объединить, ***используя флаг в программе вывода массива в файл***. Флаг определяет, куда выводится массив: в файл или на консоль. Если на консоль, то рассчитывается с каким шагом по индексам надо выводить массив (см. следующий пункт).
4. Найти в массиве B элемент, равный числу *p* с точностью 0,001, используя два алгоритма поиска, описанных в методической части лабораторной работы. Оценить время поиска.
5. Составить программу, позволяющую сортировать массив разными методами с соответствующей оценкой времени.
6. После сортировки вывести на консоль каждый *m*-й элемент массива.
7. Провести исследование не менее трёх методов сортировки, алгоритмы которых описаны *в соответствующих лекционных материалах*, и провести анализ эффективности методов.

Правила инициализации для массивов даны в следующих вариантах. Для правильной алгоритмизации методов инициализации обязательна разработка контрольного примера для одного элемента каждого массива и включения его в Отчёт.

Вариант 1. Создать целочисленный массив A из 4-х первых цифр (без округления) синусов случайных чисел в диапазоне от 0 до $\frac{\pi}{3}$. Например, $sin\frac{\pi}{8} = 0{,}006853838$ ... Тогда соответствующий ему элемент массива A = 68. Массив B – вещественный и равен косинусу его индекса, умноженному на $\sqrt{\lg Z}$, где Z – случайное число (провести контроль ОДЗ).

Вариант 2. Создать целочисленный массив A из 8-х старших битов случайных чисел, вычисляемых в заданном диапазоне. Массив B – вещественный и вычисляется по формуле $sin\frac{x}{y-lnx}$, где x и y – случайные вещественные числа. Провести контроль ОДЗ.

Вариант 3. Создать целочисленный массив A из 1-й, 2-й, 5-й и 6-й цифр косинусов случайных чисел случайных чисел в диапазоне от $-\frac{\pi}{3}$ до $+\frac{\pi}{3}$. Массив B – вещественный и инициализируется как остаток от деления двух случайных вещественных чисел.

Вариант 4. Создать целочисленный массив A из первых 5 цифр дробной части частного a / b, где a b и – случайные числа. Массив B – вещественный и инициализируется согласно алгоритму Выполните генерацию массива, используя закономерность: 0; 0.1; 0.12; 0.123;…; 0,123456789; 1; 1.1; …1.123456789;…

Вариант 5.* Создать вещественный массив A, инициализируемый из первых N_1 простых чисел, делённых на его порядковый номер. Например: A_0=1., A_1=2/2=1; A_2=3/3=1; A_3=5/4=1.25; A_4=7/5=1.4, … Алгоритмы поиска простых чисел можно посмотреть здесь https://habr.com/ru/post/468833/). Массив B – вещественный, его элементы равны разности между случайным числом и его случайным округленным с избытком или с недостатком. Например: случайное число z=3.01247 случайно округляется с избытком, тогда B[i]= 3.01247-4=- 0,98753, другое случайное число z=2,965324 случайно округляется с недостатком, тогда, соответствующий элемент массива будет B[i]= 2,965324-2=+ 0,965324.

Вариант 6. Создать целочисленный массив A, инициализируемый k_1 битами случайного целого числа x, отсчитываемыми от бита с номером k_2, где k_1 и k_2 - случайные числа, допускающие взятие битов в пределах типа числа x. Например, если *unsigned x*, то k_2 может иметь значения от 0 до 14, а k_1 от 15 до 1, соответственно. Массив B – вещественный, его элементы равны дробной части отношения двух случайных чисел C и D. Пример: C=42, D=23, A[i] = 0,826086.

* * *

4. Контрольные вопросы

1. Какие методы сортировок вы усвоили?
2. Кратко сформулируйте и напишите в отчёт результаты анализа применения методов поиска и сортировки ?
3. Перечислите все способы получения доступа к массиву, формируемому в теле другой функции.
4. Напишите прототип функции, которая сортирует массив по любому выбранному Вами способу с применением указателя на функцию.

5. Домашнее задание

Выполняется по желанию, если осталось время после выполнения основного задания. Выполнить задание по любому, ранее не решаемому, варианту лабораторной работы.