МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Кузбасский государственный технический университет имени Т. Ф. Горбачева»

Кафедра электропривода и автоматизации

Составитель
А. Н. Гаргаев

ПРОГРАММИРОВАНИЕ

Методические материалы

Рекомендованы учебно-методической комиссией направления подготовки
11.03.04 Электроника и наноэлектроника в качестве электронного издания
для использования в образовательном процессе

Кемерово 2018

Рецензенты:

Григорьев А. В. – кандидат технических наук, доцент кафедры электропривода и автоматизации

Семыкина И. Ю. – председатель учебно-методической комиссии направления подготовки 11.03.04 Электроника и наноэлектроника

Гаргаев Андрей Николаевич

Программирование: методические материалы [Электронный ресурс] для обучающихся направления подготовки 11.03.04 Электроника и наноэлектроника, профиль Промышленная электроника, очной формы обучения / сост. А. Н. Гаргаев; КузГТУ. – Электрон. издан. – Кемерово, 2018. – Систем. требования : Pentium IV; ОЗУ 8 Мб; Windows 95; мышь. – Загл. с экрана.

В методических материалах по дисциплине «Программирование» приведены темы, материалы и задачи для лабораторных занятий, самостоятельной работы, а также вопросы для самоконтроля.

© КузГТУ, 2018
© Гаргаев А. Н.,
составление, 2018

СОДЕРЖАНИЕ

Лабораторная работа №1 СТРУКТУРЫ 4
1. ЦЕЛЬ РАБОТЫ 4
2. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 4
2.1. Объявление структуры 4
2.2. Инициализация структур 6
3. ЗАДАЧИ 6
4. ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ 7
Лабораторная работа №2 СТЕКИ И ОЧЕРЕДИ 8
1. ЦЕЛЬ РАБОТЫ 8
2. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 8
2.1. Стеки 8
2.2. Реализация стека 9
2.3. Очереди 10
2.4. Реализация очереди 10
3. ЗАДАЧИ 12
4. ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ 13
Лабораторная работа №3 ВЫЧИСЛИТЕЛЬНЫЕ АЛГОРИТМЫ 14
1.ЦЕЛЬ РАБОТЫ 14
2.КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 14
2.1. Целочисленные алгоритмы 14
2.2.Многочлены 17
2.3.Последовательности и ряды 17
2.4.Численное решение уравнений 20
2.5. Численное интегрирование 24
2.6.Интерполяция 26
3.ЗАДАЧИ 29
4. ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ 30
Лабораторная работа №4 КОНЕЧНЫЕ АВТОМАТЫ 31
1. ЦЕЛЬ РАБОТЫ 31
2. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 31
3. ЗАДАЧИ 35
4. ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ 36
СПИСОК ЛИТЕРАТУРЫ 37
САМОСТОЯТЕЛЬНАЯ РАБОТА 38

Лабораторная работа №1

СТРУКТУРЫ

1. ЦЕЛЬ РАБОТЫ

1.Изучить структуры, их инициализацию и способы применения;

2.Выполнить предложенные задачи для закрепления знаний по программированию структур.

2. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1. Объявление структуры

Структура – объединение одного или нескольких объектов, в качестве которых могут выступать переменные, массивы, указатели и т.д. Структура представляет собой сложный тип данных, составленный из простых типов.

Некоторые объекты, которые используются в конкретной программе, могут быть логически связаны между собой, например: абсцисса (double*x*) и ордината (double*y*) могут являться координатами некоторой точки плоскости, или фамилия (charname[20]), год рождения (intyear) и группа (chargroup[30]) могут содержать сведения о том или ином студенте и т.д. Поэтому для удобства работы с такими объектами их можно группировать под одним именем. Если массив представляет собой набор однотипных данных, то структура является более универсальным типом данных, поскольку она может содержать объекты различных типов.

Объявление структуры осуществляется с помощью ключевого слова struct, за которым следует имя структуры, и далее идет перечисление списка объектов, заключенных в фигурные скобки.

Общая форма объявления структуры:

```
structтип_структуры
{
тип Имя_Элемента_1;
тип Имя_Элемента_2;
. . .
тип Имя_Элемента_n;
};
```

Переменные, объявленные внутри структуры, называются ее элементами. Элементы одной структуры должны иметь уникальные имена, хотя имена элементов различных структур могут содержать одинаковые имена. Каждое объявление структуры должно заканчиваться точкой с запятой.

Например, описание структуры, представляющей собой точку плоскости, выглядит следующим образом:

```
struct DOT
{
double x; /* абсцисса */
double y;   /* ордината */
};
```

Описание же структуры, которая представляет собой сведения о студенте, может выглядеть таким образом:

```
struct STUDENT
{
charname[20];   /* фамилиястудента */
intyear;/* год рождения */
chargroup[30]; /* группа */
};
```

При объявлении структуры внутри программы резервирования памяти под нее не происходит, она просто описывает новый тип данных. Объявив структуру, можно создавать переменные этого типа:

```
struct DOT A, B; /* переменные A и B типа DOT */
struct STUDENT stud;/* переменная stud типа STUDENT */
```

Этим самым мы объявили две переменные-структуры *A* и *B*, которые могут содержать координаты точек плоскости, и одну переменную-структуру *stud*. Обращение к отдельным элементам структуры происходит через операцию "точка" (.): *A.x, A.y, stud.name, stud.year, stud.group.*

Важным моментом при описании структуры является место, где она объявляется. Если объявление происходит вне всех функций (такое объявление называется внешним), тогда это объявление может использоваться всеми функциями, следующими за ним. Если же структура объявляется внутри какой-то функции, тогда это объявление может использоваться только этой функцией.

Если структура передается в качестве аргумента функции, то, в отличие от массивов, передача происходит по значению. Чтобы осуществить передачу структуры по ссылке, необходимо передать ее адрес.

Также можно создавать массивы структур. Например, можно задать последовательность из 10 элементов точек плоскости:

```
struct DOT a[10];
```

Тогда обращение к координатам точек будет происходитm следующим образом:

a[0].x, a[0].y,…, a[9].x, a[9].y

2.2. Инициализация структур

Как и массивы, структурные переменные при их определении можно инициализировать. Инициализация структуры может осуществляться двумя способами:

- присвоение значений элементам структуры в процессе объявления переменной, относящейся к типу структуры;
- присвоение начальных значений элементам структуры с использованием функций ввода-вывода (например, printf() и scanf()).

В первом способе инициализация осуществляется по следующей форме:

Struct Имя_Структуры Имя_Переменной={Значение_Элемента1, Значение_Элемента2, . . ., Значение_Элементаn};

Пример:

```
struct DOT A = {1.2, -2.5};
struct STUDENT stud = {"Иванов", 1990, "КБ-11"};
```

Второй способ инициализации объектов языка Си с использованием функций ввода-вывода.

Пример:

```
#include <stdio.h>
#include <math.h>
struct DOT
{
double x;
double y;
};
intmain( )
{
struct DOT A;
scanf("%lf", &A.x);
scanf("%lf", &A.y);
printf("d = %.2f\n", sqrt(A.x*A.x + A.y*A.y));
return 0;
}
```

3. ЗАДАЧИ

1. Написать программу сложения и умножения двух комплексных чисел.

2. Пользователь вводит данные о количестве студентов, их фамилии, имена и балл каждого. Программа должна определить средний балл и вывести фамилии и имена студентов, чей балл выше среднего.

3. Написать программу, в которой хранятся данные о товарах, их количестве и цене. При запуске программы эта информация выводится на экран. Далее пользователю должно предлагаться вводить номера товаров и их новое количество. Изменение данных должно завершаться, если пользователь вводит специально оговоренный символ (например, 0). После этого все данные о товарах должны снова выводиться на экран.

4. В программе заданы месяц и год двух дат. Пользователь вводит еще одну дату (только месяц и год). Определить, принадлежит ли третья дата диапазону от первой даты до второй включительно. Задачу решить с использованием структуры данных. Дана сторона квадрата. Найти его периметр.

5. Написать программу, которая по номеру месяца возвращает общее число дней года вплоть до этого месяца.

6. Переделайте предыдущую программу таким образом, чтобы она запрашивала у пользователя день, месяц, год и выдавала общее количество дней в году вплоть до данного дня. Месяц может обозначаться номером, названием месяца или его аббревиатурой.

7. Составьте структуру для учетной картотеки служащего, которая содержала бы следующие сведения: фамилию, имя, отчество; год рождения; домашний адрес; место работы, должность; зарплату; дату поступления на работу.

8. Составьте программу "справочник по таблице Менделеева", которая по названию химического элемента выдавала бы его характеристики. Таблицу инициализируйте массивом структур.

4. ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

1. Что такое структура?
2. Как объявляется структура?
3. Что такое элемент структуры?
4. Как происходит обращение к отдельным элементам структуры?
5. Как оформляется оператор ввода? Что можно указывать в качестве элементов списка ввода? Как работает оператор ввода (что происходит при его выполнении)?
6. Какими способами осуществляется инициализация структуры?

Лабораторная работа №2

СТЕКИ И ОЧЕРЕДИ

1. ЦЕЛЬ РАБОТЫ

1. Ознакомиться со стеками и их реализацией;
2. Изучить очереди и их реализацию;
3. Решенить предоставленные задачи для закрепления знаний об очередях и стеках.

2. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1. Стеки

Стек – это упорядоченный набор элементов, в котором добавление новых и удаление существующих элементов допустимо только с одного конца, который называется вершиной стека.

Стек называют структурой типа LIFO (LastIn – FirstOut) – последним пришел, первым ушел.

Стек похож на стопку с подносами, уложенными один на другой – чтобы достать какой-то поднос надо снять все подносы, которые лежат на нем, а положить новый поднос можно только сверху всей стопки. На рисунке показан стек, содержащий 6 элементов.

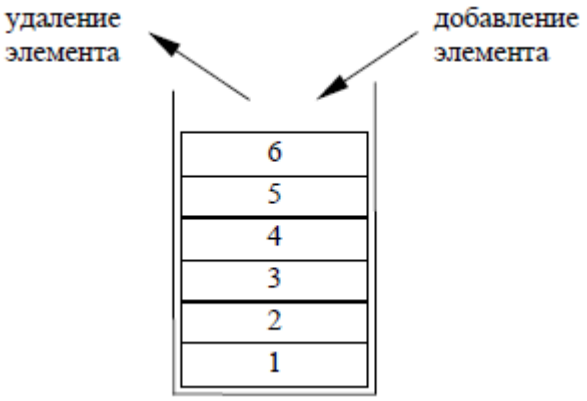

Рис. 1. Пример стека

В современных компьютерах стек используется для

• размещения локальных переменных;

• размещения параметров процедуры или функции;

• сохранения адреса возврата (по какому адресу надо вернуться из процедуры);

• временного хранения данных.

На стек выделяется ограниченная область памяти. При каждом вызове процедуры в стек добавляются новые элементы (параметры, локальные переменные, адрес возврата). Поэтому если вложенных вызовов будет много, стек переполнится.

Над стеком выполняются следующие операции:

1) добавление в стек нового элемента (операция Push);

2) доступ к последнему включенному элементу (вершине стека);

3) исключение из стека последнего включенного элемента (операцияPop).

2.2. Реализация стека

Если максимальный размер стека заранее известен, его можно реализовать в программе в виде массива. Удобно объединить в одной структуре сам массив и его размер. Объявим новый тип данных – стек на 100 элементов-символов.

```
constint MAXSIZE = 100;
struct Stack
{
char data[MAXSIZE];
intsize;
};
```

Для работы со стеком надо определить, как выполняются две операции – добавление элемента на вершину стека (Push) и снятие элемента с вершины стека (Pop).

```
void Push ( Stack&S, char x )
{
if ( S.size == MAXSIZE )
{
printf ("Стекпереполнен");
return;
}
S.data[S.size] = x;
S.size ++;
}
```

Поскольку нумерация элементов массива data начинается с нуля, надо сначала записать новый элемент в S.data[S.size], а затем увеличить размер стека. В процедуре предусмотрена обработка ошибки «переполнение стека». В этом случае на экран будет выдано сообщение «Стек переполнен». Можно также сделать функцию Push, которая будет возвращать 1 в случае удачного добавления элемента и 0 в случае ошибки. Обратите внимание, что стек S передается в процедуру по ссылке, то есть, фактически передается адрес

этого стека в памяти. Поэтому все операции со стеком в процедуре выполняются непосредственно со стеком вызывающей программы.

```
char Pop ( Stack&S )
{
if ( S.size == 0 ) {
printf ("Стекпуст");
return char(255);
}
S.size --;
return S.data[S.size];
}
```

Функция Pop возвращает символ, «снятый» с вершины стека, при этом размер стека уменьшается на единицу. Если стек пуст, функция возвращает символ с кодом 255 (который никогда не может находиться в стеке по условию задачи и сигнализирует об ошибке).

2.3. Очереди

Очередь – это упорядоченный набор элементов, в котором добавление новых элементов допустимо с одного конца (он называется концом очереди), а удаление существующих элементов – только с другого конца, который называется началом очереди. Хорошо знакомой моделью является очередь в магазине. Очередь называют структурой типа FIFO (FirstIn – FirstOut) – первым пришел, первым ушел. На рисунке изображена очередь из 3-х элементов.

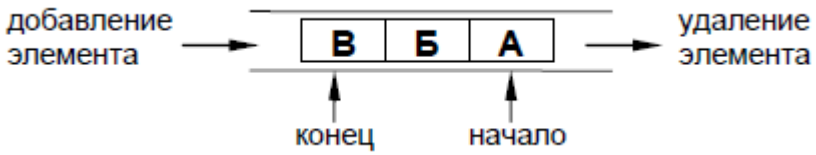

Рис. 2. Пример очереди

Наиболее известные примеры применения очередей в программировании – очередь событий системы Windows и ей подобных. Очереди используются также для моделирования в задачах массового обслуживания (например, обслуживания клиентов в банке).

2.4. Реализация очереди

Если максимальный размер очереди заранее известен, его можно реализовать в программе в виде массива. Удобно объединить в одной структуре сам массив и его размер. Объявим новый тип данных – очередь на 100 элементов (целых чисел).

```
constint MAXSIZE = 100;
struct Queue
{
int data[MAXSIZE];
intsize, head, tail;
};
```

Если у стека один конец «закреплен» (не двигается), то у очереди «подвижны» оба конца. Чтобы не сдвигать все элементы в массиве при удалении или добавлении элемента, обычно использую две переменные head и tail – первая из них обозначает номер первого элемента в очереди, а вторая – номер последнего. Если они равны, то в очереди всего один элемент. Массив как бы замыкается в кольцо – если массив закончился, но в начале массива есть свободные места, то новый элемент добавляется в начало массива, как показано на рисунках.

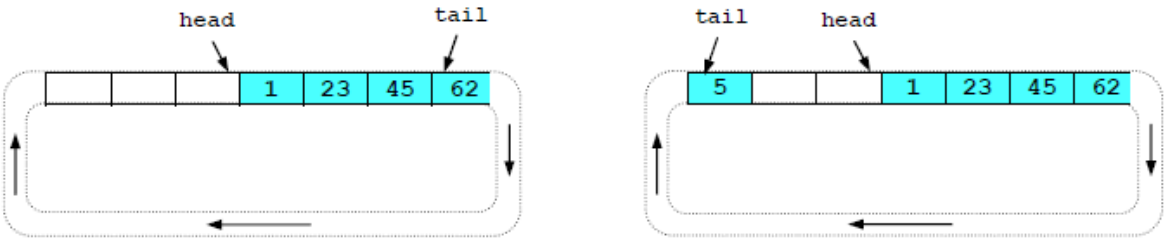

Рис. 3. Представление работы очереди

Для работы с очередью надо определить, как выполняются две операции – добавление элемента в конец очереди (PushTail) и удаление элемента с начала очереди (Pop).

```
void PushTail( Queue&Q, int x )
{
if ( Q.size == MAXSIZE )
{
printf ("Очередьпереполнена\n");
return;
}
Q.tail++;
if( Q.tail>= MAXSIZE ) // замыкание в кольцо
Q.tail = 0;
Q.data[Q.tail] = x;
Q.size ++;
}
```

Поскольку очередь может начинаться не с начала массива (за счет того, что некоторые элементы уже «выбраны»), после увеличения Q.tail надо проверить, не вышли ли мы за границу массива. Если это случилось, новый элемент записывается в начало массива (хотя является хвостом очереди). В

процедуре предусмотрена обработка ошибки «переполнение очереди». В этом случае на экран будет выдано сообщение «Очередь переполнена». Можно также сделать функцию PushTail, которая будет возвращать 1 в случае удачного добавления элемента и 0 в случае ошибки.

Обратите внимание, что очередь Q передается в процедуру по ссылке, то есть, фактически передается адрес этой структуры в памяти.

```
int Pop ( Queue&Q )
{
int temp;
if ( Q.size == 0 )
{
printf ("Очередьпуста\n");
return 32767; // сигналобошибке
}
temp = Q.data[Q.head];
Q.head ++;
if ( Q.head>= MAXSIZE )
   {
Q.head = 0;
Q.size --;
return temp;
}
}
```

Функция Pop возвращает число, полученное с начала очереди, при этом размер очереди уменьшается на единицу. Если стек пуст, функция возвращает число 32767 (предполагается, что оно не может находиться в очереди по условию задачи и сигнализирует об ошибке).

3. ЗАДАЧИ

1. Сформировать стек целых чисел, вводимых пользователем, а затем его содержимое вывести на экран, не удаляя элементы, из которых извлекается информация. Также написать функцию удаления стека.

2. Расположить элементы целочисленного массива размером n в обратном порядке с использованием стека.

3. Написать функцию, которая слова в текстовом файле распечатывает в обратном порядке. По файлу можно пройти только один раз.

4. Элементы целочисленного массива записать в очередь. Написать функцию извлечения элементов из очереди до тех пор, пока первый элемент очереди не станет четным.

5. Даны две непустые очереди, которые содержат одинаковое количество элементов. Объединить очереди в одну, в которой элементы исходных очередей чередуются.

6. Даны две непустые очереди. Элементы каждой из очередей упорядочены по возрастанию. Объединить очереди в одну с сохранением упорядоченности элементов.

7. Пусть имеется файл действительных чисел и некоторое число C. Используя очередь, напечатать сначала все элементы, меньшие числа C, а затем все остальные элементы.

4. ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

1. Что такое стековый принцип сохранения элементов?
2. Поясните структуру LIFO.
3. Как задается стек?
4. Какие основные операции реализуются для стеков?
5. Как выполняется удаление и добавление элемента в стек?
6. Приведите пример стека.
7. Что такое структура данных типа очередь?
8. Какие отличия между стеком и очередью?
9. Поясните структуру FIFO.
10. Какие основные операции используются при работе с очередью?
11. Как реализовывается очередь?
12. Приведите пример очереди.

Лабораторная работа №3

ВЫЧИСЛИТЕЛЬНЫЕ АЛГОРИТМЫ

1.ЦЕЛЬ РАБОТЫ

1. Изучить алгоритмы основных численных методов;
2. Оценить параметры и качество решения численными методами;
3.Овладеть навыками применения численных методов при решении прикладных задач.

2.КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1. Целочисленные алгоритмы

К этой группе относятся некоторые классические алгоритмы для работы с целыми числами, которые были известны еще древним математикам.

Алгоритм Евклида

Алгоритм Евклида (3 в. до н.э.) служит для вычисления наибольшего общего делителя (НОД) двух натуральных чисел. Оригинальный алгоритм Евклида основан на равенстве:

НОД(a,b) = НОД(a-b,b) = НОД(a,b-a).

Чтобы не работать с отрицательными числами, можно сформулировать алгоритм так: заменяем большее из двух чисел разностью большего и меньшего до тех пор, пока они не станут равны – это и есть НОД.

Функция для вычисления НОД может быть записана в двух вариантах:

а) рекурсивная

```
int NOD ( int a, int b )
{
if ( a == b )
   {
return a;
   }
if ( a< b )
   {
return NOD(a,b-a);
   }
else
   {
return NOD(a-b,b);
```

```
    }
}
```

б)нерекурсивная

```
int NOD ( int a, int b )
{
while ( a != b )
{
if ( a> b )
{
a -= b;
        }
else
        {
b -= a;
        }
}
return a;
}
```

Конечно, в данном случае нерекурсивная форма лучше: она работает быстрее и не расходует стек попусту.

Наименьшее общее кратное

Наименьшим общим кратным (НОК) двух чисел называется наименьшее число, которое делится без остатка на оба исходных числа.

Найти наименьшее общее кратное можно очень просто, если знать НОД исходных чисел. Действительно, пусть x = x1·d и y = y1·d, где d = НОД(x,y). Тогда

$$\text{НОК}(x, y) = xy_1 = x_1y = \frac{xy}{\text{НОД}(x, y)};$$

Для вычисления НОД можно использовать алгоритм Евклида.

Решето Эратосфена

Другая задача, которую решили математики Древней Греции – нахождение всех простых чисел в интервале от 1 до заданного числа N. Самым быстрым и поныне считается алгоритм Эратосфена (275–195 гг. до н.э.). Существует предание, что Эратосфен выписал в ряд на папирусе все натуральные числа и затем прокалывал каждое второе (то есть делящееся на 2), потом – каждое третье, и т.д. В конце этой процедуры остаются «непроколотыми» только простые числа, которые делятся только на себя и на 1.

Вместо папируса мы будем использовать массив A, в котором элемент A[i] для всех i от 1 до N принимает два возможных значения:

1, число «не проколото» и является кандидатом на простые;
0, число «проколото» и больше не рассматривается.

В целях экономии памяти можно выбрать не целые переменные (занимающие 4 байта), а символьные (например, unsignedchar, которые занимают 1 байт). Кроме того, для экономии памяти еще в 8 раз можно рассматривать отдельные биты числа, но этот вариант мы не затрагиваем.

Будем перебирать все числа k от 2 до N (включительно) и в цикле «вычеркивать» числа, кратные k: 2k, 3k, ... Очевидно, что надо использовать только те числа k, которые еще не вычеркнуты. Ниже полностью приведена программа, реализующая этот алгоритм. Заметьте также, что удобно выделить место в памяти под N+1 элемент, чтобы использовать элементы с

A[1] по A[N]. При этом элемент A[0] не используется. Можно этого избежать, но это усложнит программу.

```
#include <stdio.h>
main()
{
unsigned char *A;
inti, k, N;
printf ("Введите максимальное число");
scanf( "%d", &N );
A = new unsigned char[N+1];  // выделитьпамятьподмассив
if ( A == NULL )
{
return1;  // выход в случае ошибки
   }
for ( i = 1; i<= N; i ++ )
A[i] = 1;
for ( k = 2; k*k <= N; k ++ )
if ( A[k] != 0 )
for ( i = k+k; i<= N; i += k ) A[i] = 0;
for ( i = 1; i<= N; i ++ )
if ( A[i] == 1 ) printf ( "%d\n", i );
}
```

Главное преимущество этого алгоритма – высокая скорость работы, основной недостаток – большой объем необходимой памяти. Он демонстрирует одну из принципиальных проблем программирования: как правило, повышение быстродействия алгоритма требует увеличения необходимого объема памяти, а экономия памяти приводит к снижению

быстродействия. В реальных ситуациях чаще всего приходится идти на компромисс.

2.2.Многочлены

Как известно, многочленом (полиномом) называют функцию вида:

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0;$$

Часто требуется вычислить значения многочлена в заданной точке $x=x_0$. Если выполнять вычисления «напрямую», каждый раз возводя x в степень(с помощью умножения), требуетсяn сложений и 1+2+…+ (n-1)+ $n = \frac{n(n+1)}{2}$ умножений. Однако для решения этой задачи придумали более эффективный способ.

Схема Горнера

Представим многочлен в виде:

$$f(x) = (\ldots(a_n x + a_{n-1})x + a_{n-2})x + \ldots a_2)x + a_1)x + a_0$$

При этом для вычисления значения функции при известном x требуется всего n сложений и n умножений, что значительно быстрее. Ниже приведена функция на языке Си, реализующая схему Горнера:

```
floatGorner( floatx, floata[], intn )
{
float v = 0.;
for ( inti = n; i>= 1; i -- )
v = ( v + a[i] ) * x;
v += a[0];
return v;
}
```

2.3.Последовательности и ряды

Последовательности

Последовательность – это набор элементов, расположенных в определенном порядке.

Пока будем рассматривать только последовательности чисел. Все элементы последовательности имеют номера, обычно начиная с 1. Для того, чтобы задать последовательность, используют два следующих способа:

1. Рекуррентную формулу, которая позволяет вычислить элемент с номером n, зная один или несколько предыдущих. Например, последовательность 1, 3, 5, 7, …можно задать рекуррентной формулой $a_n = a_{n-1} + 2$.

2.Формулу для n-ого члена последовательности. Для той же последовательности легко получить $a_n = 2n - 1$. Правая часть этой формулы зависит только от номера элемента n. Для хорошо известной арифметической прогрессии формула n-ого члена имеет вид:

$$a_n = a_1 + (n - 1)d;$$

где a_1– начальный элемент, а d – разность. Для геометрической прогрессии, соответственно, $a_n = a_1 q^{(n-1)}$, где q – знаменатель прогрессии.

В таблице ниже приводятся примеры последовательностей.

Таблица 1

№	Последовательность	a_n
1	1,3,5,7, …	$2n - 1$
2	1,3,7,15, …	$2^n - 1$
3	2,9,28,65, …	$n^3 - 1$
4	$1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots$	$\frac{1}{n}$
5	$1, 1, \frac{5}{7}, \frac{7}{15}, \ldots$	$\frac{2n-1}{2^n - 1}$

Самые распространенные задачи, связанные с последовательностями:

• определить, сколько членов убывающей последовательности больше некоторого заданного значения;

• найти первый член последовательности, удовлетворяющий заданному условию;

• найти сумму заданного числа элементов последовательности или сумму элементов, удовлетворяющих заданному условию.

Задача. Для последовательности 4 в таблице найти

1) сумму элементов, которые больше 10^{-5},

2) сколько элементов вошло в эту сумму,

3) каков первый элемент, меньший 10^{-5}?

При решении подобных задач особое внимание надо обращать на эффективность вычислений и точность. Если на каждом шаге делать умножение и возведение в степень 2^n, такая программа будет неэффективной.

Можно сделать иначе: ввести две переменные, скажем, u и d, которые будут обозначать 2*n-1 и 2^n, соответственно. Первая будет с каждым шагом увеличиваться на 2, а вторая – в 2 раза. Ниже приведены две программы,

одна из которых работает в 3 раза быстрее другой. В обоих случаях искомые величины находятся в переменных s, n, a.

```
int n;
float a, s;
n = 0; s = 0;
while (1)
{
n ++;
a=(2*n-1)/(pow(2,n)-1);
if ( a< 1.e-5 ) break;
s += a;
}
```

```
int n;
float a, s, u, d;
n = 0; s = 0;
u = 1; d = 2;
while (1)
{
n ++;
a = u / (d-1.);
if (a < 1.e-5) break;
u += 2; d *= 2;
}
```

Ряды

Ряд – это сумма бесконечного числа элементов последовательности.

Конечно, на практике невозможно просуммировать бесконечное количество элементов, поэтому бесконечную сумму заменяют конечной – частичной суммой ряда, в которую входят первые k элементов:

$$S = \sum_{1}^{\infty} a_n \approx S_k = \sum_{1}^{k} a_n \, ;$$

Существует два типа рядов: расходящиеся и сходящиеся. Для расходящегося ряда частичная сумма S_k может стать больше по модулю, чем любое наперед заданное число. Примером расходящегося ряда является гармонический ряд:

$$S = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots ;$$

Для сходящегося ряда частичная сумма стремится, при увеличении количества элементов k, к некоторому фиксированному конечному значению, которое и называется суммой этого ряда.

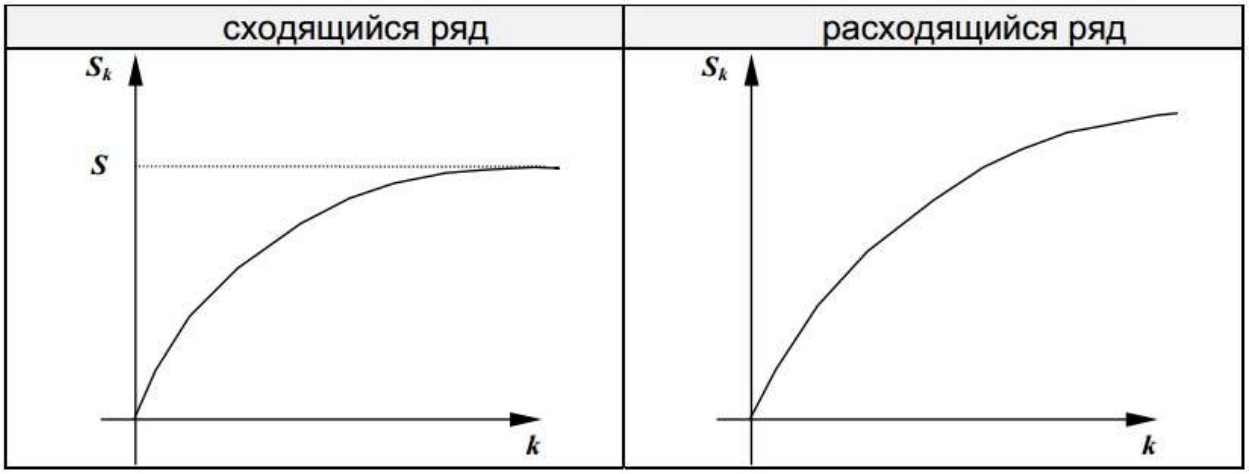

Рис. 1. Сходящийся и расходящийся ряд

Для сходящихся рядов часто можно вычислить сумму ряда с заданной точностью ε, так что

$$|S - S_k| = \left|\sum_{k+1}^{\infty} a_n\right| < \varepsilon;$$

В общем случае это достаточно сложная математическая задача, однако для рядов особого вида она имеет простое решение.

Если ряд знакопеременный и его элементы убывают, то он сходится, и ошибка в вычислении суммы не превышает модуля последнего отброшенного элемента.

Знакопеременным называется ряд, в котором знаки элементов чередуются, например

$$S = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \ldots + \frac{(-1)^{n+1}}{n} + \cdots;$$

2.4. Численное решение уравнений

Многие уравнения невозможно решить аналитически, выписав решение в виде формулы в явном виде. В таких случаях используют приближенные численные методы. Приближенными они называются потому, что мы в принципе не можем найти точное решение x^*, однако можем найти некоторое приближение к нему, которое отличается от точного решения x^* не более, чем на заданную величину ε.

Мы будем рассматривать уравнения вида f(x)=0, где f(x) – функция одной переменной. Один из численных методов решения таких уравнений – метод деления отрезка пополам или дихотомии. К его преимуществам можно отнести то, что мы можем гарантировать, что ошибка не превышает заданную величину. С другой стороны, надо заранее определить интервал, в котором находится один и только один корень.

Метод хорд

Так же, как и метод дихотомии, этот метод предназначен для уточнения корня на известном интервале [a,b] при условии, что корень существует и непрерывная функция f(x) имеет на концах отрезка разные знаки.

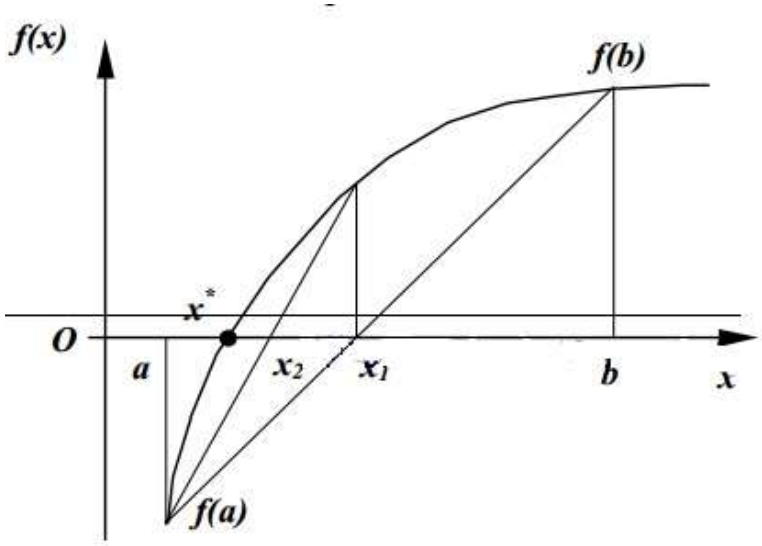

Рис. 2. Метод хорд

За следующее приближение к корню принимается не середина отрезка [a,b], как в методе дихотомии, а значение x в точке, где прямая, соединяющая точки (a,f(a)) и (b,f(b)) пересекает ось OX. Уравнение прямой, проходящей через эти точки, имеет вид:

$$\frac{x-a}{b-a}=\frac{y-f(a)}{f(b)-f(a)};$$

В точке пересечения имеем y=0. Подставляя в уравнение y=0, получаем

$$x_1=a-\frac{(b-a)f(a)}{f(b)-f(a)};$$

Далее все происходит так, как при использовании метода дихотомии: проверяем, на каком из интервалов $[a,x_1]$ и $[x_1,b]$ находится пересечение, и смещаем соответственно точку a или точку b.

Более сложен вопрос о том, когда закончить итерации. Обычно применяется один из двух критериев:

1) разность между двумя последовательными приближениями x_kи x_{k-1} стала меньше заданной точности ε, или…

2) модуль значения функции f (x_k) в точке x_k стал меньше заданного значения ε1.

При этом нельзя гарантировать, что ошибка составит не более какой-либо величины – она зависит от вида функции.

```
float Chord( float a, float b, float eps, float eps1 )
{
float x, x0 = a, fa, fb, fx;
   while (1)
{
fa = F(a); fb = F(b);
x = a - (b -a)*fa / (fb - fa);  // следующее приближение
fx = F(x);
if( fabs(fx) <eps1 ) break; // проверка условия2
if( fa*fx< 0 ) b = x;  // сужаем интервал поиска
else
a = x;
if (fabs(x-x0) <eps )break; // проверка условия1
x0 = x;
}
returnx;
}
```

Параметры eps и eps1 обозначают точность для изменения x и значения функции в этой точке. Эта функция может быть еще оптимизирована, так как можно не вычислять заново каждый раз значения f(a) и f(b)– одно из них мы знаем с предыдущего шага.

Метод Ньютона (метод касательных)

Чаще всего на практике используется метод Ньютона, который обладает быстрой сходимостью и требует знать только начальное приближение x_0 к корню, а не интервал, в котором он находится.

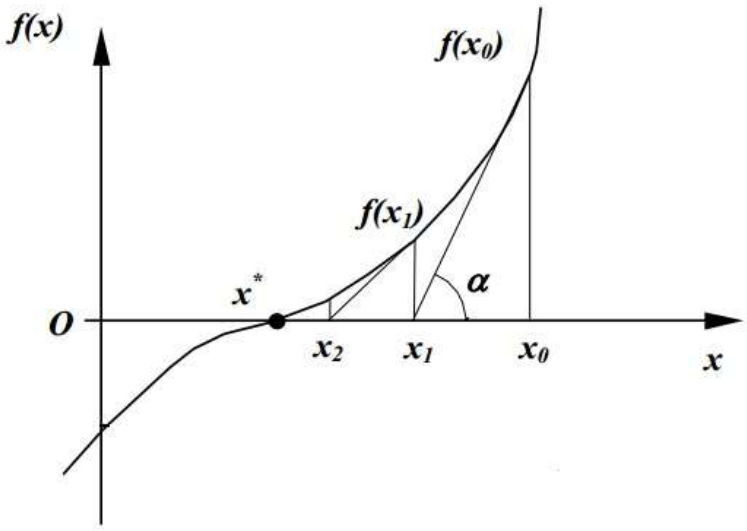

Рис. 3. Метод Ньютона (метод касательных)

За следующее приближение к корню принимается значение x в точке пересечения касательной, проведенной к кривой в точке (x_0,f(x_0)), и оси OX. Поскольку

$$\mathrm{tg}\alpha = f'(x_0) = \frac{f(x_0)}{x_0 - x_1};$$

для k-ого шага итерации получаем

$$x_k = x_{k-1} = \frac{f(x_{k-1})}{f'(x_{k-1})};$$

Метод Ньютона обладает высокой скоростью сходимости. Обычно точность 10^{-6} достигается за 5-6 итераций. Его главный недостаток в том, что надо на каждом вычислять производную функции, а выражение для нее может быть неизвестно, например, если она задана таблицей. Иногда используют модифицированный метод Ньютона, в котором на всех шагах используется производная, вычисленная в точке x_0. Важно также правильно выбрать начальное приближение к корню, иначе метод Ньютона может «зациклиться».

Приведенная ниже функция использует функции F(x) и DF(x), которые возвращают соответственно значение функции и ее производную в точке x.

```
float Newton (float x0, float eps)
{
float f1;
do
{
f1 = F(x0) / DF(x0);
x0 -= f1;
}
while ( fabs(f1) > eps );
```

```
returnx0;
}
```

Метод Ньютона используется также для решения систем нелинейных уравнений с несколькими переменными, но формулы оказываются более сложными, и мы их не рассматриваем.

2.5. Численное интегрирование

Численное интегрирование – вычисление значения определённого интеграла (как правило, приближённое). Численное интегрирование применяется, когда:

- подынтегральная функция не задана аналитически (например, она представлена в виде массива значений в узлах некоторой расчётной сетки);
- аналитическое представление подынтегральной функции известно, но аналитическое выражение ее первообразной затруднено.

Основная идея большинства методов численного интегрирования состоит в замене подынтегральной функции на более простую, интеграл от которой легко вычисляется аналитически. Наиболее часто с этой целью используется метод трапеций.

Пусть требуется вычислить определенный интеграл:

$$S = \int_a^b f(t)dt;$$

Как известно, определенный интеграл представляет собой площадь под кривой, ограниченной подынтегральной функцией. Для вычисления определенного интеграла площадь под кривой аппроксимируется прямоугольными трапециями.

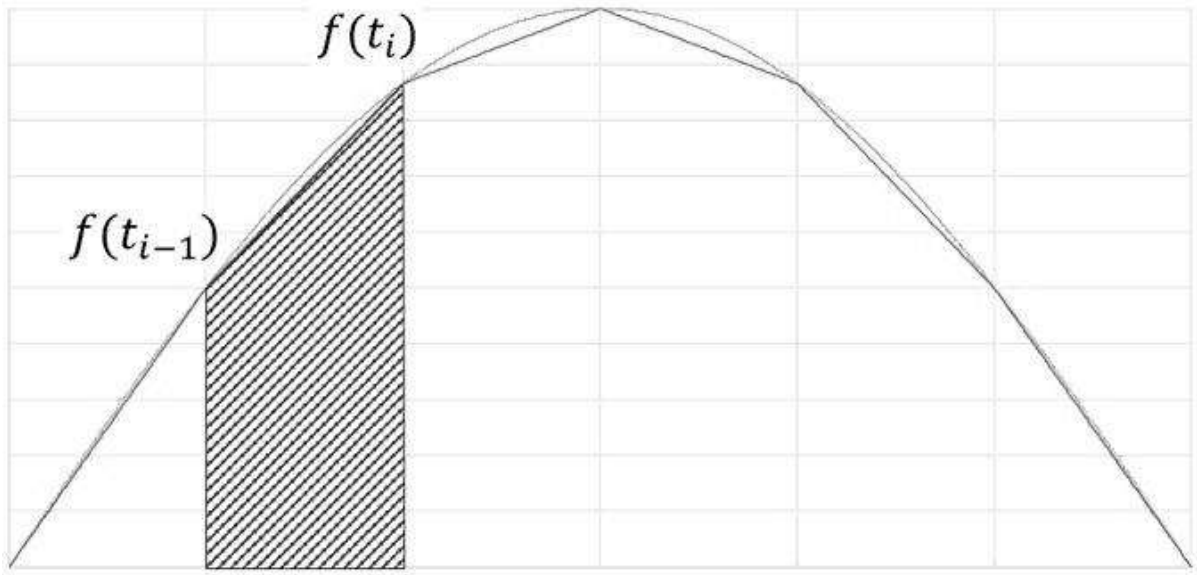

Рис. 4. Метод трапеций

Площадь каждой составляющей прямоугольной трапеции можно определить, как

$$s_i = \frac{f(t_i) + f(t_{i-1})}{2} \cdot (t_i - t_{i-1});$$

Высота трапеций составляет небольшую величину такую, что на отрезке

$$\Delta t = t_i - t_{i-1};$$

функцию f(t) можно заменить линейной.

При этом площадь под графиком функции можно вычислить как сумму площадей составляющих прямоугольных трапеций.

$$S = \sum_{i=a}^{b} s_i = \sum_{i=a}^{b} \frac{f(t_i) + f(t_{i-1})}{2} \cdot (t_i - t_{i-1});$$

Если высота трапеций принята постоянной, то значение интеграла можно вычислить как

$$S = \left(\frac{f(t_a) + f(t_b)}{2} + \sum_{i=a+\Delta t}^{b-\Delta t} f(t_i) \right) \cdot \Delta t;$$

В случае если величина Δt достаточно мала и количество дискретных точек, в которых определяется значение функции f(t) достаточно велико, указанную выше формулу с допустимой погрешностью можно заменить приближенной

$$S \approx \sum_{i=a}^{b} f(t_i) \cdot \Delta t;$$

Реализация:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define NUMPOINT 300 // количествоточек
#definePI 3.1415926536
// Функциявычислениянопределенногоинтеграла
double Integral(double *f, double step)
{
double value = 0;
for (inti = 0; i< NUMPOINT; i++)
{
value += f[i];
}
```

```
value *= step;
return value;
}
intmain()
{
double *f;
double step, t;
 double S;
inti;
system("chcp 1251");
system("cls");
f = (double*)malloc(NUMPOINT * sizeof(double));
printf("Количество точек = %d\n", NUMPOINT);
step = PI / NUMPOINT; // величина шага (высота трапеций)
printf("Величина шага = %lf\n", step);
t = 0.0;
// Инициализация значений функции f(t)=sin(t)
for (i = 0; i<NUMPOINT; i++)
{
f[i] = sin(t);
t += step;
}
S = Integral(f, step); // вычислениеинтеграла
printf("Значениеинтеграла = %lf", S);
getchar();
return 0;
}
```

Результат выполнения:

Количество точек=3000
Величина шага=0.001047
Значение интеграла=2.000000

Как известно:

$$\int_0^{\pi} \sin(t)\, dt = 2\ ;$$

2.6. Интерполяция

Интерполяция, интерполирование – в вычислительной математике способ нахождения промежуточных значений величины по имеющемуся дискретному набору известных значений.

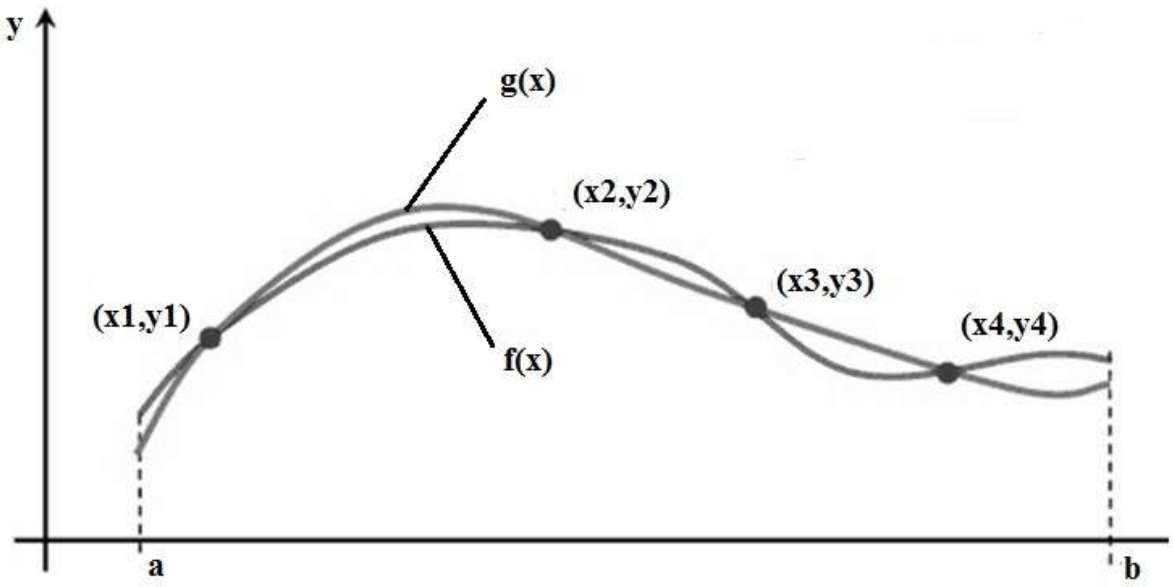

Рис. 5. Интерполяция

Многим из тех, кто сталкивается с научными и инженерными расчётами, часто приходится оперировать наборами значений, полученных опытным путём или методом случайной выборки. Как правило, на основании этих наборов требуется построить функцию, на которую могли бы с высокой точностью попадать другие получаемые значения. Такая задача называется аппроксимацией. Интерполяцией называют такую разновидность аппроксимации, при которой кривая построенной функции проходит точно через имеющиеся точки данных.

Существует много конечно-разностных методов интерполяции. Наиболее распространенным является метод Ньютона для интерполяции «вперед» (метод Ньютона-Грегори). Интерполяционный полином в этом случае имеет вид:

$$P_n(x) = C_0 + C_1(x - x_0) + C_2(x - x_0)(x - x_1) + \cdots + C_n(x - x_0)(x - x_1) \dots (x - x_{n-1});$$

Коэффициенты C находим по формуле:

$$C_0 = y_0;$$

$$C_0 = \frac{y_1 - y_0}{h} = \frac{\Delta y_0}{h};$$

$$C_2 = \frac{y_2 - 2y_1 + y_0}{2h^2} = \frac{\Delta^2 y_0}{2h^2};$$

$$C_j = \frac{\Delta^j y_0}{j!\, h^j};$$

Реализация программы:

```
using System;
namespace Interpolation
{
class Program
   {
      private static double step;
      private static double[] X = { 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
      8.0, 9.0, 10.0 };
      private static double[] Y = { -1.0, 2.0, 17.0, 50.0, 107.0, 194.0, 317.0,
      482.0, 695.0, 962.0, 1289.0 };

      static void Main(string[] args)
      {
         step = X[1] - X[0];
         double point = 6.5;
Console.WriteLine("Test function at point " + point +
         " has value " + TestFunction(point));
Console.WriteLine("Newton method at point " + point +
         " has value " + TestFunction(point));
Console.ReadLine();
      }

      // Function y = x^3 + 3x^2 - x - 1
      static double TestFunction(double x)
      {
         double result = Math.Pow(x, 3);
         result += 3 * Math.Pow(x, 2);
         return result - x - 1;
      }
      static double Pn(double x)
      {
int n = X.Length;
         double result = Cj(0);
         for (int j = 1; j < n; j++)
         {
            double something = Cj(j);
            for (int k = 0; k < j; k++)
            something *= x - X[k];
            result += something;
         }
         return result;
      }
```

```
        static double Cj(int j)
        {
            if (j == 0)
                return Y[0];
            double delta = Delta(j, j);
            return delta / (Factorial(j) * Math.Pow(step, j));
        }

        static double Delta(int p, inti)
        {
            if (p == 1)
                return Y[i] - Y[i - 1];
            return Delta(p - 1, i) - Delta(p - 1, i - 1);
        }

        static double Factorial(double value)
        {
            if (value == 0)
                return 1;
            return value * Factorial(value - 1);
    }
    }
    }
```

3.ЗАДАЧИ

1. Вычислить значение выражения, учитывая ограниченность входных данных: $y=\frac{\sqrt{x^2-4}}{x-8}$; $z=\frac{1-tgx}{1+tgx}$

2. Вычислить значение функции в точке:

$$f(x)=\begin{cases} x^2, \text{если } k=0 \\ \sin(x), \text{если } k=1, \text{в остальных случаях функция не определена} \\ 0, \text{если } k=2 \end{cases}$$

3. Во введенном с клавиатуры натуральном числе подсчитать количество цифр 2 в его десятичной записи.

4. Даны два целых числа. Найти:

а) их среднее арифметическое;

б) их среднее геометрическое.

5. Составить программу обмена значениями трех переменных величин a, b, c по следующей схеме:

а) b присвоить значение c, a присвоить значение b, c присвоить значение a;

б) b присвоить значение a, c присвоить значение b, a присвоить значение c.

6. Даны n целых чисел. Определить, сколько среди них отрицательных.

7. Среди введенных с клавиатуры десяти натуральных чисел найти максимальное четное число.

8. Дана последовательность целых чисел заканчивающаяся 0. Найти произведение целых чисел, которые одновременно больше 10 и кратны 4.

9. Дана непустая последовательность натуральных чисел, за которой следует ноль. Определить порядковый номер первого числа, являющегося палиндромом (при этом ноль во внимание не принимать). Если таких чисел нет, то сообщить об этом. Палиндром – это число, не изменяющееся при выписывании его цифр в обратном порядке.

10. Дана последовательность из n целых чисел. Определить, встречаются ли в этой последовательности два подряд идущих одинаковых числа.

11. Найти значение интеграла на отрезке [a,b]. Количество отрезков вводим с клавиатуры. Включить в программу точное значение интеграла (самому посчитать), найти погрешность. На печать вывести примерное значение, точное и погрешность. Функция: x*exp(x)*sin(x). Отрезок: [0,1].

4. ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

1. Напишите второй закон Ньютона в виде дифференциального уравнения. Какой физический смысл имеет коэффициент перед производной в этом уравнении?

2. Напишите все производные первого и второго порядка функции четырех переменных f (t, x, y,z)

3. Оценить точность получения значения определенного интеграла методом прямоугольников.

4. Способы уменьшения погрешности нахождения определенного интеграла методом трапеций.

5. Как можно повысить точность интерполяции?

6. Можно ли использовать метод Лагранжа для экстраполяции?

7.Как определить погрешность интерполяции в узле?

8. Проанализируйте погрешность решения методом Ньютона в зависимости от выбранного шага.

9. Сколько различных интерполяционных полиномов можно построить для n заданных точек?

10. Как влияет количество узлов интерполяции на точность интерполяции?

Лабораторная работа №4

КОНЕЧНЫЕ АВТОМАТЫ

1. ЦЕЛЬ РАБОТЫ

1. Ознакомиться с назначением конечных автоматов;
2. Изучить принципы работы конечных автоматов;
3. Получение практических навыков построения КА на основе заданной регулярной грамматики;
4. Решение предоставленных задач и приобретение навыков программирования в среде Си.

2. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

В задачах, обрабатывающих ввод посимвольно, или череду каких-нибудь событий, часто возникают похожие решения, которые включают в себя некоторое текущее состояние-переменную и правила перехода между переменными после обработки нового ввода или события. Такие конструкции были обобщены теоретиками под названием конечного автомата.

Конечный автомат состоит из конечного множество состояний S, начального состояния S0 и конечного множества переходов. Переходы имеют вид (d,X,t,a),

где d∈S – начальное состояние, из которого возможно выполнить переход;

X – множество символов, которые активируют переход;

t∈S –состояние, в которое будет выполнен переход

a – действие, которое конечный автомат выполняет при переходе из состояния d в t. Часто действие может отсутствовать.

Конечный автомат, это некоторое обобщение системы, которая в зависимости от внешних событий может изменять своё состояние, и в процессе таких изменений как-то реагировать на эти события. Частым примером конечного автомата является кофейный автомат. У автомата есть различные события, на которые он может реагировать (нажатия кнопок автомата), состояние, которое он может изменять при нажатии кнопок, действия, которые происходят при изменениях состояний.

Рассмотрим конечный автомат на примере программы, которая подсчитывает во вводимом тексте количество слов, состоящих только из заглавных букв. Основная идея в том, чтобы следить, были ли все уже введённые буквы текущего слова заглавными, и сохранять эту информацию в переменной, которая называется “состояние”. Состояний у нас будет три: после ввода одного или нескольких пробелов, все введённые буквы текущего слова заглавные, и *не все* введённые буквы слова заглавные.

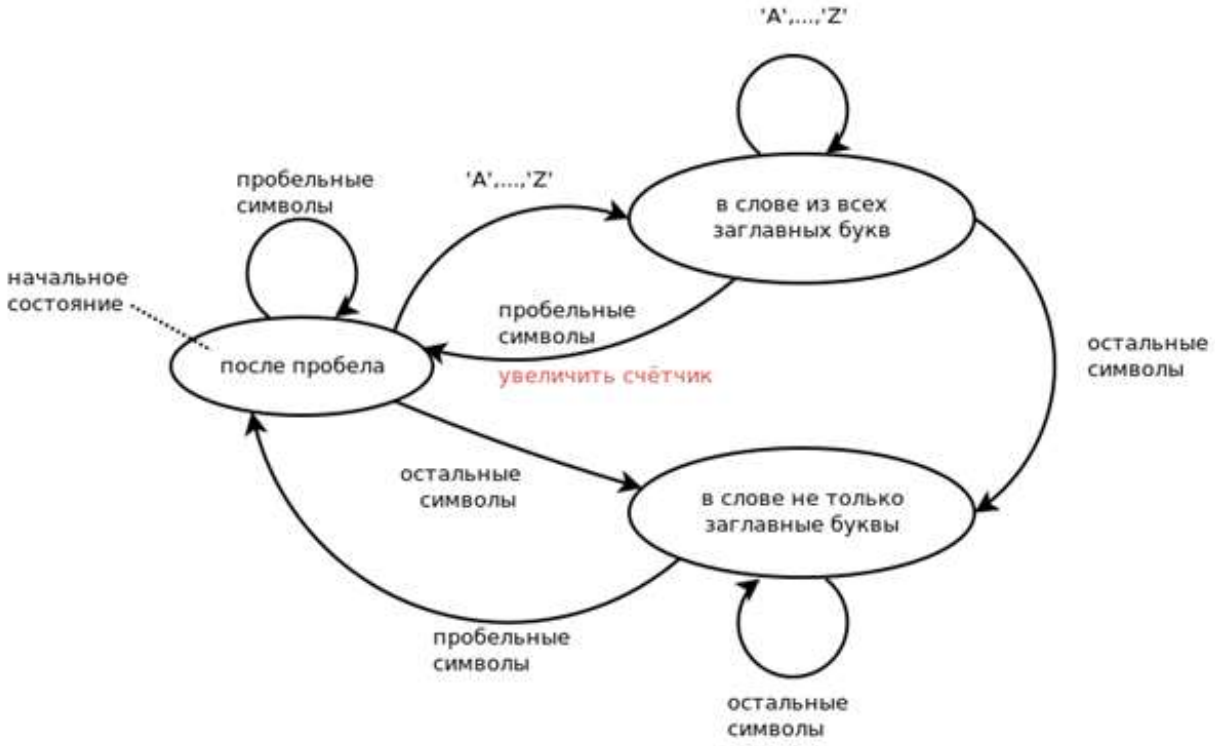

Рис. 1. Схема, поясняющая принцип работы конечного автомата

Эту схему нетрудно перенести в код на Си. После считывания символа со стандартного ввода мы сначала проверяем, в каком состоянии мы находимся, затем, в зависимости от состояния и поступившего символа, совершаем переход в другое состояние или остаёмся в текущем. Обратите внимание на использование конструкции if ... elseif. Так писать в данном случае обязательно, чтобы случайно не обработать один введённый символ в двух разных состояниях.

Код программы:

```
#include<stdio.h>
// Воспользуемся функциями isspace и isupper из ctype.h,
// isspace возвращает истину, если символ является
// пробельным (' ', '\t', '\r', '\n'), и ложь в противном случае,
// isupper возвращает истину, если символ является
// буквой в верхнем регистре, и ложь в противном случае.
#include<ctype.h>
// Определим по константе для каждого состояния
#define SPACES 0
#define ALLCAPITAL 1
#define NOTALLCAPITAL 2
intmain()
{
int c;
int state = SPACES;
intcapital_count = 0;
   while ((c = getchar()) != EOF) {
```

```
        if (SPACES == state) {
            if (isspace(c)) {
// просто не меняем текущее состояние
            } elseif (isupper(c)) {
state = ALLCAPITAL;
            } else {
                state = NOTALLCAPITAL;
            }
        } else if (ALLCAPITAL == state) {
if (isspace(c)) {
// Увеличиваем счётчик, затем переходим
                // в начальное состояние.
++capital_count;
state = SPACES;
            } else if (isupper(c)) {
// просто не меняем текущее состояние
            } else {
state = NOTALLCAPITAL;
            }
        } else if (NOTALLCAPITAL == state) {
            if (isspace(c)) {
                state = SPACES;
} else {
                // просто не меняем текущее состояние
            }
        }
    }
    // Если в конце мы находимся в слове из больших букв,
    // то, так как пробела уже не случится, увеличим
    // счётчик в отдельном if-е.
if (ALLCAPITAL) {
        ++capital_count;
    }
printf("Words with all catpital letters: %d\n", capital_count);

return 0;
}
```

Мы уже часто встречались с конструкциями вида:

```
int x;
x = get_some_value();
if (0 == x) {
// Действие 0.
} elseif (1 == x) {
```

```
   // Действие 1.
} else if (2 == x) {
// Действие 2.
} else if (3 == x) {
// Действие 3.
} else {
   // Действие для всех остальных x.
}
```

Такие участки кода часто возникают в программах, предоставляющих пользователю некоторое меню из действий, и выполняющих выбранное действие. В этом случае x было бы номером выбранного пользователем действия. Также мы только что ознакомились с конечными автоматами, при реализации конечного автомата x было бы переменной, хранящей состояние конечного автомата.

Приведённый выше код отлично работает, но если x равно 3, то будет выполнено четыре сравнения чисел: x с нулём, с единицей, с двойкой и тройкой. Но дело в том, что в процессоре для таких случаев, когда мы сравниваем целое число с некоторым набором констант, начинающихся с нуля, существует специальная оптимизация, называемая таблицей переходов. Её суть в том, что компилятор загоняет указатели на машинный код первой инструкции в каждой ветке указанного оператора в массив (таблицу переходов), а затем использует x для выбора нужного перехода из массива. При этом, если x выходит за границу правильных индексов таблицы переходов, то выбирается ветвь по умолчанию (последний else в примере выше).

Для использования этой оптимизации в Си есть оператор switch:

```
int x;
x = get_some_value();
switch(x) {
case 0:
   // Действие 0.
break;
case 1:
   // Действие 1.
break;
case 2:
   // Действие 2.
break;
case 3:
// Действие 3.
break;
default:
// Действие для всех остальных x.
break;
```

}

Из описанного выше происхождения оператора switch следуют некоторые его ограничения:

- в switch(x) x должно быть переменной целого типа или выражением с результатом целого типа;
- в case могут стоять только целые числа, и они должны быть константами. Иначе при компиляции не получится сформировать таблицу переходов.

Кроме того, стоит обратить внимание на break в конце каждого действия. Если его не поставить, то произойдёт переход к первой инструкции следующего действия. break в конце ветки default, которая выполнится, если не подошёл ни один case, поставлен просто для аккуратности. Отсутствие break является одной из частых ошибок. Чтобы не забывать break, рекомендуется писать его сразу же, как только написан case, а уже потом добавлять в case действия.

3. ЗАДАЧИ

1. Строка состоит из букв, цифр и пробелов. Программа считает количество десятичных чисел (“1a2” – не число).
2. Строка состоит из букв, цифр и пробелов. Программа считает количество восьмеричных чисел (“1a2” – не число).
3. Строка состоит из букв, цифр и пробелов. Программа считает количество слов (“ф2ы” – не слово).
4. Строка состоит из букв и пробелов. Программа считает количество слов, начинающихся с заглавной буквы.
5. Строка состоит из букв и пробелов. Программа считает количество слов из заглавных букв.
6. Строка состоит из букв и пробелов. Программа считает количество слов из строчных букв.
7. Строка состоит из букв и пробелов. Программа считает количество слов из трех букв.
8. Строка состоит из букв и пробелов. Программа считает количество слов из двух букв.
9. Строка состоит из букв и пробелов. Программа считает количество слов из двух одинаковых букв.
10. Строка состоит из букв, цифр и пробелов. Программа считает количество четных чисел.
11. Строка состоит из букв, цифр и пробелов. Программа считает количество нечетных чисел.
12. Строка состоит из букв, цифр и пробелов. Программа считает количество шестнадцатеричных чисел.
13. Строка состоит из букв, цифр и пробелов. Программа считает сумму чисел.

4. ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

1. Что такое конечный автомат (КА)?
2. Как можно задать КА?
3. Расскажите о том, как функционирует КА.
4. Приведите пример простого конечного аппарата.

СПИСОК ЛИТЕРАТУРЫ

1. Златопольский Д. М. Сборник задач по программированию. – 3-е изд., перераб. и доп. – Санкт-Петербург: БХВ-Петербург, 2011. – 304 с.: ил. – (ИиИКТ)

2. Карпов Ю. Г. Теория автоматов: учебник для вузов. – Санкт-Петербург: Питер, 2002. – 224 с.

3. Солдатенко И. С. Основы программирования на языке Си: учеб. пособие. – Тверь: Твер. гос. ун-т, 2017. – 159 с.

4. Рацеев С. М. Программирование на языке Си: учебное пособие / С. М. Рацеев. – Ульяновск: УлГУ, 2015. – 323 с.

САМОСТОЯТЕЛЬНАЯ РАБОТА

Общие положения

Целью самостоятельной работы по дисциплине «Программирование» является систематическое изучение дисциплины в течение семестра, закрепление и углубление полученных знаний и навыков, подготовка к предстоящим занятиям, приобретении новых знаний и умений для углубления профессиональных компетенций. Самостоятельная работа студента осуществляется путем выполнения домашних заданий в форме конспектирования, составления обзоров.

Содержание самостоятельной работы

Домашнее задание № 1.

Изучение темы «Составные типы данных». Составить конспект по изученному материалу.

Домашнее задание № 2

Подготовка к лабораторной работе №1

Домашнее задание № 3

Изучение темы «Структура данных: очередь, стек». Составить конспект по изученному материалу.

Домашнее задание № 4

Подготовка к лабораторной работе №2

Домашнее задание № 5

Изучение темы «Структура данных: связанный список». Составить конспект по изученному материалу.

Домашнее задание № 6

Подготовка к лабораторной работе №3

Домашнее задание № 7

Изучение темы «Конечный автомат с памятью». Составить конспект по изученному материалу.

Домашнее задание № 8

Подготовка к лабораторной работе №4

Промежуточная аттестация проходит в форме зачета. Правила проведения зачета приведены в рабочей программе дисциплины.

ВОПРОСЫ К ЗАЧЕТУ

1. Структуры. Определение, объявление, инициализация, примеры.
2. Стеки. Определение, структура LIFO, основные операции, примеры.
3. Стеки. Определение, структураLIFO, инициализация, примеры.
4. Очереди. Определение, структура FIFO,основные операции, примеры.
5. Очереди. Определение, структура FIFO, реализация, примеры.

6. Перечислите различия между стеком и очередью.

7. Алгоритм Евклида. Описание, назначение, варианты вычисления НОД.

8. Решето Эратосфена. Описание, назначение, преимущества и недостатки метода.

9. Схема Горнера. Описание метода нахождения значения многочлена в заданной точке.

10. Последовательности. Определение, задачи, связанные с последовательностями, примеры.

11. Ряды. Определение, типы рядов, сумма ряда, знакопеременные ряды.

12. Метод Хорд. Описание, принцип работы, реализация метода, критерии конца итераций.

13. Метод Ньютона. Описание, принцип работы, реализация.

14. Напишите второй закон Ньютона в виде дифференциального уравнения. Какой физический смысл имеет коэффициент перед производной в этом уравнении?

15. Численное интегрирование, метод трапеции. Описание, применение, принцип работы, реализация.

16. Способы уменьшения погрешности нахождения определенного интеграла методом трапеций.

17. Интерполяция. Описание, принцип работы, реализация.

18. Способы повышения точности интерполяции?

19. Конечные автоматы. Определение, принцип работы, задание конечного автомата, пример.