Министерство образования Российской Федерации
РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ГИДРОМЕТЕОРОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

А.Д. Шишкин
Е.А. Чернецова

ПРАКТИКУМ

Программирование на языке Си

Рекомендовано в качестве учебного пособия
Редакционно-издательским советом РГГМУ

Санкт-Петербург
2003

УДК 681.3.657.1

Шишкин А.Д., Чернецова Е.А. Практикум «Программирование на языке Си». – СПб.: изд. РГГМУ, 2003. – 52 с.

Рецензент А.И. Яшин, проф. ГЭТУ (ЛЭТИ)

В практикум включены девять лабораторных работ, которые охватывают все разделы дисциплины «Методы и средства программирования». Лабораторные работы ориентированы на приобретение студентами навыков программирования на языке Си в пакете Borland C++.

Практикум предназначен для студентов обучающихся по специальностям «Морские информационные системы и оборудование» и «Информационная безопасность телекоммуникационных систем», а также может быть полезным для всех желающих ознакомиться с основами программирования на языке Си.

ISBN 5-86813-045-6

Предисловие

Дисциплина «Методы и средства программирования» является обязательной для студентов, обучающихся на кафедре «Морские информационные технологии», и ее цель – обеспечение фундаментальной подготовки студентов по использованию методов и средств программирования на одном из языков.

Главной задачей дисциплины является развитие навыков у студентов по формализации научных и инженерных задач, выбора методов решения, навыков составления алгоритмов и программ на языке Си.

В лабораторный практикум включены лабораторные работы, которые охватывают основные разделы дисциплины «Методы и средства программирования» Работы составлены таким образом, что практическое освоение языка идет по возрастанию сложности решаемых задач. В первой работе студенты знакомятся со средой программирования Borland C++, осваивая методы составления, редактирования, компиляции, тестирования и обработки программ.

Остальные работы следуют в том порядке, в котором традиционно изучаются соответствующие разделы в курсе программирования, начиная с линейных процессов и заканчивая графикой. В каждой работе приводятся краткие сведения по теме, методика программирования типовых процессов и индивидуальные задания для работы.

Содержащиеся в практикуме сведения по теории программирования, методические указания и рекомендации по выполнению лабораторных работ позволяют использовать его в качестве учебного пособия для закрепления курса лекций. Знания и навыки, полученные при изучении дисциплины, позволят грамотно и эффективно использовать вычислительную технику, решать на ней прикладные задачи как в процессе обучения, так и в последующей профессиональной работе в подразделениях Гидрометслужбы, Госкомэкологии и других ведомствах.

Лабораторная работа № 1

Освоение работы в интегрированной среде программирования

Цель работы: приобретение практических навыков работы в интегрированной среде (ИС) программирования Borland C.

Назначение и режимы работы ИС

ИС программирования Borland C предназначена для создания, редактирования и запуска в работу программ, написанных на языке программирования Си и C++. В ней пользователь получает ряд услуг, упрощающих процесс создания и отладки программ. В системе имеется развитая система помощи, которая позволяет получить справочную информацию по всем вопросам программирования. Она предоставляет такие возможности как многооконный режим работы, поддержка работы с мышью, возможность быстрого перехода к другим программам и быстрого возврата, наличие макроязыка редактора и др.

Система имеет два режима работы. Первый, наиболее важный, который используется практически всегда – это режим работы с интегрированной средой TURBO. В ней работа осуществляется с помощью меню. Второй режим работы – использование традиционного метода, когда в начале применяется какой-либо текстовый редактор для создания текстового файла с программой, затем, набирая в командной строке DOS соответствующие команды для компиляции, компоновки и, наконец, выполнения программы.

Запуск среды TURBO

Войти в ИС программирования можно двумя путями:

1) найти на рабочем столе ярлык системы Turbo C++ и щелкнуть по нему левой кнопкой мыши два раза;
2) в среде NC найти каталог TURBO C++ , в подкаталоге BIN выбрать команду

bc.exe

и нажать клавишу <Enter>.

После загрузки ИС она представляется как графический образ, состоящий из трех компонентов: строки меню, оконного пространства, строки состояния.

Выход из системы

Чтобы окончательно покинуть систему можно воспользоваться командой QUIT в меню FILE или нажать комбинацию клавиш ALT+X. Для временного выхода в операционную систему (чтобы выполнить какую-либо команду DOS), оставив при этом программу в памяти машины, используется опция DOS Shell меню FILE. Для возврата в систему требуется набрать в командной строке DOS команду Exit.

Работа с окнами

ИС является многооконной. Главную роль играют окна редактора, но используются также и окна других видов: справочной системы, диалога, контроля, сообщений, наблюдений, проектов и вывода.

Каждое окно имеет свой номер. Переход в другое окно (когда работа ведется с несколькими окнами) осуществляется нажатием клавиш ALT+N, где N – цифра, определяющая номер окна.

Операции с окнами могут выполняться с помощью меню, мыши, либо с помощью «горячих» клавиш. Описание команд меню приведено в[2].

Задание на выполнение работы

1. Загрузите интегрированную среду Turbo C++. Ознакомтесь с окнами и пунктами главного меню. Откройте новое окно с помощью пункта меню File/New.

В появившееся окно наберите демонстрационную программу (задается преподавателем). Набор текста осуществляется обычным набором средств, знакомых вам по работе с текстовыми редакторами. Запишите набранную программу под своим оригинальным именем в каталог, указанный преподавателем. Для этого выберите в меню команду File/Save as. В появившемся окне наберите путь, имя файла и нажмите клавишу <Enter>.

2. Откройте следующее окно. Используя директивы редактора для работы с блоками текста, выделите часть текста программы. Для этого установите курсор в начало блока текста и, используя директиву Ctrl+KB, выделите начало блока, а затем, установив курсор в конец блока текста директивой Ctrl+KK – конец блока. Сразу же после выделения текст высветится в инверсном изображении. С вы-

деленным блоком текста можно осуществлять операции копирования, перемещения, удаления и т. д.

Скопируйте выделенный блок в подготовленное вами пустое окно редактора. Для этого следует выбрать пункт меню Edit/Copy, с помощью которого выделенный текст будет скопирован в карман. Затем следует перейти в нужное окно редактора и, используя пункт меню Edit/Paste, вставить текст из кармана в окно на позицию, указанную курсором.

Отмените выделение текста директивой Ctrl+KH (повторное выполнение директивы снова выделит текст).

3. Повторите описанный выше процесс копирования с различными участками исходного текста для закрепления навыков копирования. Попробуйте переместить фрагменты текста в выбранную вами позицию в режиме копирования и переноса. Удалите часть текста, а затем вставьте его.

Выделение блока происходит также в том случае, если при нажатой клавише Shift нажимаются клавиши: стрелки перемещения курсора и клавиши Home, End, PgUp, PgDn. Кроме команд редактирования, удобно использовать следующие директивы для работы с выделенными блоками текста.

Директива	Функция
Ctrl+Del	удаление выделенного блока,
Ctrl+Ins	копирование выделенного блока в карман,
Shift+Del	перемещение выделенного блока в карман,
Shift+ Ins	вставка выделенного блока из кармана.

Проведите выделение текста и манипуляции с текстом с использованием указанных клавиш.

4. Использование мыши для работы с текстом. Использование мыши значительно упрощает работы с окнами и текстом. Для перехода из одного окна в другое необходимо щелкнуть левой кнопкой мыши в площади нужного окна. Изменить размеры окна можно «протаскиванием» правого нижнего угла окна. Перемещение окна осуществляется «протаскиванием» в нужное место поля заголовка.

Расположите с помощью мыши два окна таким образом, чтобы они занимали одинаковую площадь и располагались вертикально по всей длине экрана.

Для фиксации курсора в нужной позиции установите в нее указатель мыши и нажмите ее левую кнопку.

Протаскивание мыши с нажатой левой кнопкой приводит к такому же выделению блока текста, как и использование рассмотренных ранее директив.

Повторите действия выделения, копирования и перемещения текста с использованием мыши и меню Edit.

5. Компиляция и выполнение программы. Откройте окно с исходной программой. Если программа подверглась модификации, то загрузите ее снова с помощью меню File/Open. Для компиляции программы необходимо, чтобы текст находился в активном окне. Компиляция осуществляется с помощью пункта меню Compile и может производиться в режимах Compile, Make, Build. Компиляция в режиме Compile завершается созданием файла с расширением .obj , а компиляция в режимах Make и Build заканчивается формированием файла с расширением .exe (выполняемый файл).

Откомпилируйте программу. Если программа не содержит ошибок, то она сразу же может быть запущена на выполнение. В случае наличия ошибок, в нижнем окне будут выданы сообщения об ошибках. Здесь указывается номер строки и характер ошибки. Исправьте допущенные ошибки. (Если таковых нет, искусственно задайте и исправьте).

6. Запуск в работу программы. Выберите меню Run и откройте его подменю. Выбор команды Run приведет к немедленному выполнению всей программы. Сначала осуществляется компиляция программы в режиме Make , а затем ее запуск на выполнение. Нажатие клавиш Ctrl+F9 из активного окна вызывает аналогичные действия.

Выполнение программы приводит к активизации окна программы, в котором вы видите результаты ее работы. Для возвращения в окно программы необходимо нажать клавиши Alt+F5. Повторное нажатие клавиш вернет вас в окно редактора. Убедитесь в этом.

Для отладки программы удобно использовать пошаговое выполнение команд программы. Для этого в меню Run выбрать пункт Step Over. Нажимая клавишу F8, можно по шагам выполнить программу. Убедитесь в этом.

7. Модификация программы. В соответствии с указанием преподавателя модифицируйте исходную программу. После модификации проведите повторную ее компиляцию и запуск на выполнение. При необходимости устраните допущенные во время редактирования ошибки.

Модифицированная программа должна быть записана под оригинальным именем в каталог, созданный для группы, а также выведена на принтер (опция File/Print) и представлена в отчете.

Содержание отчета

1. Краткое изложение назначения и возможностей ИС Turbo C++.
2. Эскиз экрана с указанием основных компонентов (полей).
3. Раскрыть назначение пунктов главного меню.
4. Распечатка текста программы.

Контрольные вопросы

1. Назовите основные пункты подготовки и решения задачи в среде Turbo C++.
2. Поясните назначение основных пунктов меню Edit.
3. Раскройте назначение функциональных и «горячих» клавиш при работе с программой.

Лабораторная работа №2

Базовые операции языка Си

Цель работы: приобретение навыков программирования линейных процессов. Освоить функции ввода/вывода данных, оператора присваивания.

Краткие сведения и инструкции

Программы с линейной структурой являются простейшими и используются, как правило, для реализации простых вычислений по формулам. Программа представляет собой последовательную запись инструкций, выполняемых одна за другой.

Алгоритм программы с линейной структурой может быть представлен в виде схемы, показанной на рис.2.1

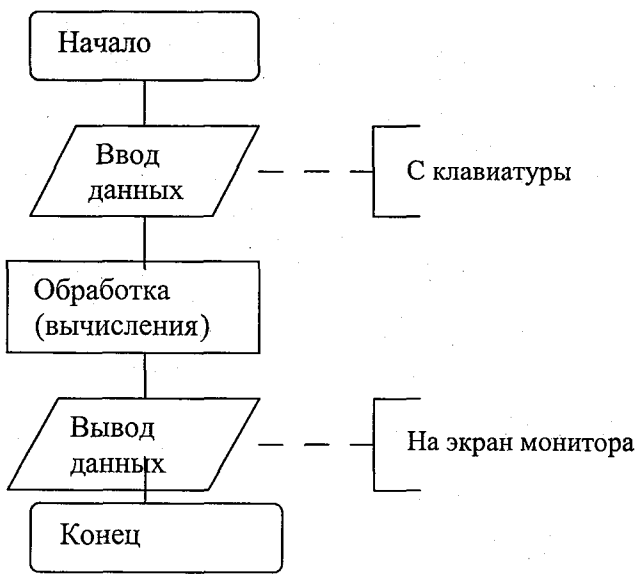

Рис. 2.1

При составлении программы, в первую очередь, необходимо определить исходные данные и объявить типы переменных. Объявления типов переменных обычно помещают в начале программы, вслед за ее

заголовком, снабжая инструкцию объявления кратким комментарием о назначении переменной. Инструкция объявления записывается так:

Тип Имя_Переменной;

Объявление переменной можно совместить с ее инициализацией. В этом случае объявлений переменной записывают следующим способом:

Тип Имя_Переменной= Начальное_значение;

В последнем выражении знак "=" обозначает инструкцию присваивания. Она предназначена для изменения значения переменных, в том числе и для вычислений по «формуле». В отличие от других языков программирования в Си и С++ инструкция присваивания, выполняющая некоторое действие, может быть записана несколькими способами. Например, вместо x = x + dx можно записать x + = dx, а вместо i= i+1 воспользоваться оператором инкремента и записать i++.

Вывод информации и сообщений на экран монитора обеспечивает функция printf(). Первым параметром этой функции является строка вывода, определяющая выводимый текст и формат вывода значений переменных, имена которых указаны в качестве остальных параметров функции.

Для ввода исходных данных с клавиатуры предназначена функция scanf(), первым параметром которой является управляющая строка, остальные параметры – адреса переменных, значения которых были введены. (Использование имени переменной, а не ее адреса в качестве параметра функции scanf() является типичной ошибкой начинающих программистов).

Для иллюстрации действия указанных операций рассмотрим следующий пример.

*Пример №1:*Введем программу, печатающую на экране: "Сейчас 2001 год".

```
#include<stdio.h>
/* пример к лаб. работе №1*/
main ( )
{
int year, month;
year = 2001;
printf ("Сейчас % d год\n", year);
}
```

Рассмотрим эту программу. Первая строка #include <stdio.h> подключает стандартный файл ввода/вывода языка Си. Это так называемый заголовочный файл (header files). В файле stdio.h находится информация о стандартной функции вывода printf (), которую мы используем.

Вторая строка /*Пример...*/ – комментарий. Строка main() – определяет имя функции. Строка: { – содержит открывающую скобку, обозначающую начало тела функции main(). Строка: int year, month; объявляет (декларирует) переменные year и month и сообщает компилятору, что они целые (int).

Строка: year = 2000; является оператором присваивания.

Строка: printf ("сейчас %d год \ n", year); является вызовом стандартной функции printf(), которая выводит на экран некоторую информацию. Она состоит из двух частей: имени функции printf() и двух её аргументов "сейчас %d год \ n" и year, разделённых запятой. Первый аргумент называется управляющей строкой (Control String). Она может содержать любые символы или спецификации формата, начинающиеся с символа %. Спецификация %d указывает, что будет выводиться целое число. Комбинация символов "\n" сообщает функции printf() о необходимости перехода каретки на новую строку.

Последняя строка программы "}" содержит закрывающую фигурную скобку тела функции main().

Если программа не содержит ошибок, то после компиляции и выполнения её на экране появится сообщение: Сейчас 2001 год. Если была допущена ошибка, то после компиляции, строка, в которой эта ошибка была допущена, будет подсвечена. Переход к предыдущей ошибке осуществляется комбинациями клавиш ALT+F7, а к последующей ALT+F8.

Пример №2. Требуется вычислить длину окружности.

Для вычисления нам потребуется ввести данные с клавиатуры, и мы будем использовать библиотечную функцию scanf(), которая позволит вводить данные во время выполнения программы.

```
/* ПРОГРАММА*/
#include <stdio.h>
/*Вычисление длины окружности */
main ()
{
```

```
int radius;
float length;
printf ("Введите значение радиуса :\n");
scanf ("%d", &radius);
length = 3.1415*2*radius;
printf ("Радиус–%d\n длина – %f\n," radius, length);
}
```

Отличие этого примера от предыдущего заключается в том, что:

1) объявлены две переменные разных типов: radius – типа целое (int); length – типа с плавающей точкой (float);
2) используется функция scanf() для ввода с клавиатуры значения радиуса окружности.

Первый аргумент функции scanf()– "%d" указывает, что будет вводиться целое десятичное число, второй аргумент указывает имя переменной, которой будет присвоено введённое значение.

В языке Си имеются специальные унарные и бинарные операторы, из которых наиболее хорошо известны положительное приращение (++) и отрицательное приращение (—), позволяющие с помощью единственного оператора добавить 1 или вычесть 1 из любого значения. Сложение и вычитание допускаются в середине выражения. Кроме того, можно задавать операции приращения до, и после вычисления самого выражения.

Пример №3.

```
sum=a+b++;
sum=a+++b;
```

Первый оператор суммирует a и b, затем результат присваивает sum, и потом значение b увеличивается на 1. Во втором выражении b увеличивается на 1, затем суммируются a и b, и далее результат присваивается sum.

Задание на выполнение работы

1) Ввести программу примера №1, провести её компиляцию и выполнение.
2) Ввести программу примера №2, провести отладку и запустить на выполнение.

3) Модифицировать программу примера №2 таким образом, чтобы она вычисляла длину окружности и площадь круга для любых радиусов.
4) Ввести программу и проверить действие операторов присваивания:

```
#include <stdio.h>
main ( )
{
 int a, b, sum;
char *format;
format="a=%d b=%d sum=%d\n";
 a=b=5;
sum=a+b printf(format, a, b, sum);
sum=a+++b; printf (format, a, b, sum);
sum=++a+b; printf (format, a, b, sum);
sum=—a+b; printf (format, a, b, sum);
sum=a—+b; printf (format, a, b, sum);
sum=a+b; printf (format, a, b, sum);
}
```

Отчёт должен содержать:

1) Тексты выполняемых программ.
2) Результаты расчётов.

Лабораторная работа № 3

Организация циклических вычислений

Цель работы: Изучить формы изменения хода выполнения программы по условию или без него; формирование циклических процессов вычислений.

Операторы и конструкции организации циклов

Для программирования вычислительных процессов с известным числом повторений обычно используют оператор цикла. Циклы необходимы, когда повторяются одни и те же вычисления до тех пор, пока выполняется некоторое условие. В языке Си известно три вида операторов цикла**: for, while, do- while.**

Основная форма цикла for имеет следующий вид:

for(инициализация; проверка условия; изменение) оператор;

На самом деле в общем виде:

for(выражение_1; выражение_2; выражение_3) оператор;

В простейшем виде инициализация используется для присвоения начального значения параметру цикла. Проверка условия – обычное условное выражение, определяющее, когда цикл будет завершен. Изменение (приращение) обычно используется для изменения параметра цикла каждый раз при повторении цикла. Как только условие становится ложным, начинает выполняться следующий за циклом оператор.

Простейший примеры оператора for:

1) *for(i=0; i<10; i++) printf ("%d \ n", i);*

В результате выполнения этого оператора будут напечатаны цифры от 0 до 9 в столбик.

2) for(i=9; i>=0; i– –) printf("%d \n",i);

Будут напечатаны цифры от 9 до 0.

В качестве параметра цикла необязательно использовать целочисленный счетчик. Приведем фрагмент программы, выводящий на экран буквы русского алфавита:

```
unsignet char ch;
for( ch='А' ; ch<= 'Я' ; ch++) printf( "%c", ch);
```

Следующий фрагмент программы

```
for( ch='o'; ch!=' N' ; ch++) scanf ("%c", &ch);
```

будет выполняться до тех пор, пока с клавиатуры не будет введен символ 'N'.

Заметим, что место, где должно стоять приращение, может оказаться пустым. Случайно или намеренно может получиться цикл, из которого нет выхода, это так называемый бесконечный цикл.

Приведем три примера таких циклов.

```
for( ; ;) printf (" Бесконечный цикл\n" );
for( i=1; 1 ; i++) printf (" Бесконечный цикл \n");
for( i=10; i>6; i++) printf (" Бесконечный цикл \n");
```

Для избежания "зацикливания" программы используется оператор break. Если оператор break встречается в составном операторе цикла, то происходит немедленное прекращение выполнения цикла и начинается выполнение следующего оператора программы.

Пример 1.

```
for( ; :)
{ ch=getchar(); /* Прочитать символ */
if (ch= ='Q') break; /* Проверка символа */
printf( "%ch", ch); /* Печатать символ */ }
```

В этом цикле будут печататься введенные символы до тех пор, пока не будет введен символ 'Q'.

Следующий оператор цикла – это цикл while. Основная его форма

while (условие) оператор;

где оператор может быть простым, составным или пустым оператором. *Условие,* как и во всех других операторах, является просто выражением. Цикл выполняется до тех пор, пока условие принимает значение *истинно*. Когда же условие принимает значение *ложно*, программа передает управление следующему оператору программы. Так же как и в цикле for в цикле while сначала проверяется условие, а затем выполняется оператор. Это так называемый цикл с *предусловием*.

В отличие от двух предыдущих циклов в операторе цикла *do-while* условие проверяется в конце оператора цикла. Форма этого оператора следующая

do{
последовательность операторов
} while(условие);

Фигурные скобки не обязательны, если внутри них находится один оператор, но для лучшей читаемости их лучше ставить.

Оператор цикла *do-while* называется оператором цикла с *постусловием.*

Какое бы условие не стояло в конце оператора, набор операторов в фигурных скобках один (первый) раз выполняется обязательно. А в циклах for и while оператор может не выполниться ни разу.

Вложенные циклы. Когда один цикл находится внутри другого, то говорят, что это вложенные циклы. Часто такие циклы встречаются при заполнении таблиц, перемножении матриц и т.д.

Пример 2. Составить программу печати таблицы умножения целых чисел.

```
# include <stdio.h>
/*Пример перемножения чисел от 1 до 10*/
main()
{
int i,j;
for( i=1;i<10 ;i++)
   { for(j=1; j<5; j++)
   printf("% d*%d=%2d",i,j,i*j);
printf("\n"); }
}
```

Прерывание выполнения циклов может быть выполнено также с помощью двух следующих операторов: continue и goto. Если оператор continue встретился в операторе цикла, то он передает управление на начало следующей итерации цикла. В циклах while и do-while – на проверку условия, в цикле for- на приращение.

Для использования оператора goto надо ввести метку (label). Метка– это идентификатор, за которым следует двоеточие.

Пример 3. Использование оператора continue.

```
#include<stdio.h>
main()
{ int i;
 for (i=1; i<100; i++)
 { if(i%7) continue;
 printf("%8d", i);
}}
```

Эта программа печатает натуральные числа, кратные семи.

Пример 4. Использование оператора goto.

```
for()
{while() {
        for() {
.....
goto exit: } } }
exit: printf ("Быстрый выход из вложенных циклов");
```

Пример 5. Вычислить конечную сумму

$$S=\sum_{i=1}^{n} \ln(kx)/k^2.$$

Общее слагаемое суммы выражается формулой

$$ak = \ln(kx)/k*k.$$

Программа имеет следующий вид:

```
#include <stdio.h>
#include<math.h> /*Подключение математической библиотеки */
/*Вычисление суммы */
main()
{
float x, sum; /*Описание типов переменных */
int k, n, k1;
printf ("Введите число x\n");
scanf("%f",&x);
printf("Введите число n\n");
scanf("%d",&n);
sum=0;
```

```
for(k=1; k<=n; k++)
{ k1=k*k;
 sum+=log(k*x)/k1;
 printf("k=%d сумма=%f\n", k, sum);}
}
```

Задание на выполнение работы

1. Ввести программу примера 5, осуществить ее отладку и работу.

2. Составить программу и произвести вычисления для одного из следующих вариантов:

1) $S = \sum_{k=1}^{5} (x+1)k \cdot \cos(x/k)/(k+1)!, \qquad x = \pi/2$

2) $S = \sum_{k=1}^{5} (\sin x)^k / k!, \qquad x = \pi/4$

3) $S = \sum_{k=2}^{8} (\cos x)^k / k!, \qquad x = \pi/4$

4) $S = \sum_{i=1}^{7} (\sin x)^i / i!, \qquad x = \pi/4$

5) $S = \sum_{i=2}^{7} (e^x - i)/(i+1)!, \qquad x = 1.0$

6) $S = \sum_{i=1}^{20} (e^{x/i} + e^{-x/i})/(i+2)^4, \qquad x = 1.0$

7) $S = \sum_{i=1}^{10} \ln(x \cdot i)/(i+2)!, \qquad x = 2.0$

8) $S = \sum_{i=-5}^{5} x^{i+5} /(i+7)!,$ $x = 2.0$

9) $P = \prod_{i=5}^{10} x^i / i!,$ $x = 5.0$

10) $P = \prod_{i=5}^{10} (x+i)^{i+1} /(i-2),$ $x = 4.0$

11) $P = \prod_{i=1}^{N} (1 + \frac{1}{\sqrt[4]{i}}),$

12) $S = \sum_{i=1}^{N} \prod_{j=1}^{i} \frac{j!}{i!}$, N<=5

13) $S = \sum_{k=m}^{N} k^2 \ln(k)$, N>m

14) $S = \sum_{i=1}^{N} \frac{i!}{(N+1)!},$

15) $F = \frac{M!+N!}{(M+N)!}$

16) $S = \sum_{i=1}^{N} \sum_{j=1}^{i} \sin(0,1*i+0,2*j);$ N=4;

17) $S = \sum_{i=1}^{N} \sum_{k=0}^{i} (i+k)^2,$

18) $S = \sum_{k=1}^{N} \frac{(-1)^{k+1}}{k(k+1)},$

19) $S = \sum_{i=1}^{N} \frac{x_i}{1 + y_i}$ при $x_i = (1+i)$, $y_i = 1/i$;

20) $S_1 = \sum_{i=1}^{N} (\sin x)^i$ $S_2 = \sum_{i=1}^{N} \sin x^i$,

21) $S = \sum_{k=1}^{N} a_k$, где $a_k = \sqrt{x^2 + \sin^2(kx)/4}$,

N – натуральное, а x – вещественное число.

22) $P = \prod_{k=1}^{N} (1 - \frac{f^2}{2k+1})$, N – натуральное; f – вещественное;

23) $S = \sum_{k=2}^{N} k^2 \ln(1 + k^2)$; N>2;

24) $S = \sum_{i=1}^{N} \sum_{k=1}^{M} \sin(\frac{\pi}{4} i + \frac{\pi}{8} k^2)$,

25) $S = \sum_{i=1}^{N} \prod_{k=1}^{N} \frac{i + x}{k}$; N– натуральное; x– вещественное;

26) $S = \sum_{k=N-5}^{N} (k - \ln(1 + 2k))$; N>5

27) $P = \prod_{i=1}^{N} \frac{1}{\sum_{k=0}^{I} (f + k)}$; N – натуральное; f – вещественное;

28) $S = \sum_{k=1}^{N} \frac{\sum_{i=1}^{N} \sin(0.01 * k * i)}{k!}$;

29) $S = \sum_{k=i}^{N} (k^3 * \sum_{i=1}^{k} (k - i^2))$;

30) $S = \sum_{i=1}^{N} (-1)^{i+1} \frac{x^{2i}}{i!}$; N – натуральное; x– вещественное.

Примечание: При вычислении факториалов необходимо помнить о максимально возможном целом числе, которое можно поместить в разрядной сетке.

Содержание отчета

1. Краткое содержание цели и задачи применения циклических процессов вычислений.
2. Алгоритм вычисления заданного преподавателем математического уравнения.
3. Программу вычислений.
4. Распечатку результатов.

Контрольные вопросы

1. Назовите основные операторы циклических процессов.
2. Назовите основные параметры цикла.
3. Как образуется бесконечный цикл и как выйти из него?

Лабораторная работа №4

Условные операторы и операторы выбора

Цель работы: изучение трех форм управления процессом выполнения программ:

1) выполнение последовательности операторов;

2) выполнение определенной последовательности операторов до тех пор, пока некоторое условие истинно;

3) использование проверки истинности условия для выбора между различными, возможными способами действия.

Основное задание:

1. Составить программу решения квадратного уравнения вида:

$$AX^2 + BX + C = 0 \qquad (1)$$

с полным анализом возможных решений (дискриминант D<0, D=0, D>0) на основе конструкций if и if-else.

2. Разработать диалоговую программу, позволяющую получать решения квадратного уравнения (1) при различных значениях коэффициентов A, B, C, а также выхода из программы по запросу, используя конструкции while или do-while (по выбору).

3. Разработать диалоговую программу, позволяющую в зависимости от значений коэффициентов получать то или иное решение, используя оператор выбора:

а) если A = 0; B и C не равны нулю;

б) если B = 0; A и C не равны нулю;

в) если C = 0; B и A не равны нулю;

г) другие возможные сочетания коэффициентов A, B, C.

Программа должна вычислять как действительные, так и комплексные корни.

4. Вывести на экран сведения об авторе, исходные значения коэффициентов, значение дискриминанта и результаты решения.

Рекомендации по программированию

1. При выполнении п.1 основного задания необходимо использовать:

- файл заголовка math.h, который позволит Вам вычислить корень квадратный из дискриминанта (sqrt(D));

- конструкцию вида:

if (выражение)

оператор, используемый, если выражение истинно.

Пример:

// подразумевается, что комплексных корней нет

if (D>0)

printf (“Решения нет \n”);

При необходимости в комбинации с if можно использовать ключевое слово else, позволяющее выполнить альтернативный оператор, или блок операторов, если условие неистинно.

Пример:

```
if (D<0)
printf (“Решения нет \n”)
else
printf (“Решение есть \n”);
```

Операторы if и else могут быть вложенными.

2. При выполнении п. 2 основного задания необходимо использовать:

дополнительно к предыдущей программе конструкцию вида:

while (выражение) оператор.

Ключевое слово while позволяет выполнять оператор или блок до тех пор, пока условие не перестанет быть истинным. Оператор или тело блока, связанного с while, не будет выполнять, если выражение изначально ложно.

Пример:

```
char vych='d';
:
:
while(vych= ='d')
{
:
            : // группа операторов
printf("Продолжать решение? (d/n)\n");
cscanf("%c",vych);
}
```

В этом фрагменте программы блок команд будет выполняться до тех пор, пока символьной переменной не будет присвоено значение 'd'.

3. При выполнении п. 3 основного задания необходимо использовать дополнительно к предыдущей программе конструкцию вида:

```
switch ( выражение_1)
{
case константа 1: оператор или группа операторов блока 1;
break;
:
:
case константа I: оператор или группа операторов блока I;
break;
default: оператор или группа операторов;
}
```

При выполнении оператора switch сначала вычисляется значение *выражения_1*, стоящего в скобках оператора switch. Тип значения должен быть одним из целых: char, int, unsigned int, long int и long unsigned. Вычисленное значение сравнивается со значениями констант операторов case.

При совпадении значения *выражения_1* с i-й константой выполняется оператор или группа операторов i-го блока. Затем управление передается на следующий (после switch) оператор, если в i-й ветви присутствует оператор break.

Если значение *выражения_1* не совпало ни с одной из констант, выполняется оператор или группа операторов, помеченных default. При ее отсутствии выполняется следующий после switch оператор.

Пример:

```
int var;
:
:
if (A1=0)
var=1;
else
 var=2;
```

```
:
 :
 switch(var)
{
case 1: {printf ("Решение уравнения \n"); …
 x1 = ……;
x2 = ……;
 :
 break;
 case 2: {printf ("Решение уравнения \n"); …
x1 = ……;
:
 :
break;
default: puts (" Ошибка \a \n");
}
```

Если значение D<0 (корни комплексные), то предусмотрите изменение знака D перед вычислением квадратного корня, иначе будет зафиксирована ошибка.

4. Выведите результаты расчетов и распечатайте программу.

Содержание отчета

1. Постановка задачи.
2. Формализация задачи.
3. Структурные схемы алгоритмов решения задачи.
4. Распечатки текстов программ с комментариями.
5. Ответы на контрольные вопросы.
6. Выводы по работе.

Контрольные вопросы

1. Какие управляющие средства выполнения программ существуют в языке Си? Объясните их синтаксис.
2. Объясните назначение всех функций, использованных в программах.
3. Какие операции используются в управляющих средствах выполнения программ.
4. Объясните различия между циклами while и do … while.
5. Для каких целей используется оператор switch ?

Лабораторная работа № 5

Обработка двумерных массивов

Цель работы: изучить принципы работы различных управляющих структур (while, do… while, for) при работе с массивами данных.

Основное задание

Составить программу работы с матрицей, для этого:

– организовать ввод элементов a[i][j] матрицы M[N][N], где i – номер строки, j– номер столбца, N –размер матрицы (N<6); ввод матрицы осуществить двумя вариантами : вручную с клавиатуры и с помощью датчика случайных чисел;

– выполнить преобразование матрицы, одним из следующих вариантов.

1. Транспонировать матрицу.
2. Возвести матрицу в квадрат.
3. Умножить к-ю строку исходной матрицы на сумму элементов главной диагонали.
4. Прибавить к к- му столбцу i-й столбец, умноженный на заданное число.
5. Прибавить к i-й строке k -ю строку.
6. Поменять местами два заданных столбца.
7. Поменять местами две заданные строки.
8. Поменять местами заданный столбец и заданную строку.
9. Сосчитать сумму элементов матрицы.
10. Сосчитать сумму элементов каждой строки матрицы.
11. Сосчитать сумму элементов каждого столбца матрицы.
12. Рассчитать среднее значение элементов матрицы.
13. Заменить знаки элементов матрицы на противоположные.
14. Сосчитать количество отрицательных элементов матрицы.
15. Рассчитать средние значения отрицательных и положительных элементов матрицы.
16. Сосчитать сумму элементов матрицы, лежащих ниже главной диагонали.
17. Сосчитать сумму элементов матрицы, лежащих выше главной диагонали.
18. Найти обратную матрицу.

19. Вычислить определитель матрицы.
20. Вычислить ij- минор матрицы.
21. Найти след матрицы.

Организовать вывод на экран дисплея сведения об авторе, номер варианта, результаты выполнения работы.

Дополнительное задание

Дополните Вашу программу функциями, определяющими активное текстовое окно.

Рекомендации по программированию

1. В начале программы предусмотрите ввод значений количества строк, столбцов матрицы M[N][N].

Пример:

```
m1: cprintf ("Введи значение количества строк и столбцов матрицы \n");
cscanf ("%d", &N);
```

В случае, когда N<2 или N>6, вывести сообщение об этом и повторить ввод N.

Пример:

```
if (N<2||N>6)
{
cprintf ("Неправильно введено значение N \n");
goto m1;
}
```

где оператор goto m1 обозначает безусловный переход к оператору с меткой m1.

Затем определите характер заполнения матрицы: автоматическое или ручное. Это можно сделать с помощью операторов:

printf (" Определите характер заполнения: 1- ручное, 2- автоматическое"); и

switch(w), где w равно 1 или 2. Для генерации псевдослучайных чисел в качестве элементов матрицы воспользуйтесь функцией rand(), инициализированной директивой препроцессора

```
#define RND ((float) rand () /32768.0)
.....
x[ i ][ j ] =RND*( целое число);
```

2. Для ввода элементов матрицы M[N][N] следует использовать конструкцию вида:

for (инициализация_1; условное выражение_1; конечное выражение_1)

for (инициализация_2; условное выражение_2; конечное выражение_2)

оператор или блок операторов // двойной цикл.

В круглых скобках содержится три выражения, разделенные символом точка с запятой. Первое из них служит для инициализации счета. Она осуществляется только один раз, когда цикл for начинает выполняться. Второе выражение – для проверки условия (она производится перед каждым возможным выполнением цикла). Когда выражение становится ложным (или в общем случае равным нулю), цикл завершается. Третье выражение вычисляется в конце каждого выполнения тела цикла.

Пример:

```
for ( i=0; i<N; i++)
for(j=0; j<N; j++)
cscanf("%f", &M[i][j]);
```

В операторе цикла можно опустить одно или более выражений (но нельзя опустить символы точка с запятой). Необходимо только включить в тело цикла несколько операторов, которые в конце концов приведут к завершению его работы.

Пример:

```
ans=2;
for(n=3;ans<=15;)
{
ans=ans*n;
cprintf("ans=%f"\n", ans);
}
```

3. Для создания диалогового режима в программе можно применить конструкции операторов цикла: while; do- while. Операторы while и do- while рассмотрены в предыдущей работе.

Пример:

```
                    char answer; int N;
do
{
:
/*группа операторов */
printf ("Хотите завершить задачу ?");
answer= getch();
}
while ( answer != 'Y' || answer != 'y');
printf ("Сделано %d – раз вычислений \n",N);
```

В этом случае выход из цикла будет осуществлен при вводе с клавиатуры (операторы getch) букв Y или y.

4. Для выполнения п.3 основного задания воспользуйтесь частью результатов, полученных в работах 3 и 4.

Содержание отчета

1. Постановка задачи.
2. Схема алгоритмов.
3. Распечатки текстов программы с комментариями.
4. Ответы на контрольные вопросы.
5. Вывод по работе.

Контрольные вопросы

1. Назовите операторы цикла, использованные Вами в программе.
2. Назовите операторы выхода из цикла и поясните их назначение.
3. Опишите примененный Вами алгоритм преобразования матрицы.

Лабораторная работа № 6

Функции пользователя и динамическое распределение памяти

Цель работы: Выработка навыков разработки функций и компоновки программы из нескольких функций (файлов) на примерах решения типовых задач линейной алгебры (операций с массивами). Знакомство с методами обработки динамических массивов.

Основное задание

Составить текст программы, состоящей из функций, реализующих операции над матрицами произвольного размера, представленных в виде динамических массивов, в соответствии с вашим вариантом задания. Для этого необходимо:

1. Познакомиться с функциями input_mat() и output_mat(), предназначенных для ввода и вывода соответственно элементов матриц произвольного размера.
2. Разработать функции, осуществляющие набор матричных операций в соответствии с вашим вариантом.
3. Дополнить функцию main() необходимыми операторами и операндами.

Рекомендации по программированию

Рассмотрим пример функции, которая вводит с клавиатуры матрицу размером m x n : m – количество строк, n – количество столбцов, выделяет необходимую область памяти для размещения ее элементов и осуществляет ввод их значений в память ПЭВМ.

```
int ** input(int m, int n)
{
int i,j;
int **a;
//Выделение динамической памяти для элементов матрицы
a= (int**)malloc(m*sizeof(int*));
for(i=0; i<m; i++)
{
a[i]=(int*)malloc(n* sizeof(int*));
for(j=0;j<n;j++)
{
a[i][j]=0;
}
}
```

```
for(i=0; i<m; i++)
for(j=0;j<n; j++)
{
printf("\nВведите элемент матрицы A(%d,%d)" ,i+1,j+1);
scanf("%d",&a[i][j]);
}
return a;
}
```

Для динамического выделения свободной памяти в данном фрагменте программы используется функция malloc(). Она возвращает указатель типа void. Для правильного использования значение этой функции надо преобразовать к указателю на соответствующий тип. При успешном выполнении операция malloc() возвращает указатель на первый байт свободной памяти требуемого размера. Если достаточного количества памяти нет, то возвращается значение 0 (нулевой указатель). Чтобы определить количество байт, необходимых для переменной, используют операцию sizeof.

Для освобождения динамической памяти используют функцию free с прототипом void *free(void *p); где *p – указатель на первый байт выделенной памяти. Прототипы обеих функций находятся в заголовочном файле STDLIB.H.

Динамическое распределение памяти удобно тогда, когда заранее неизвестно количество используемых переменных (в нашем случае – это m и n).

Для выделения динамической памяти можно так же использовать функцию

new тип [количество элементов].

Рассмотрим теперь функцию, осуществляющую вывод матрицы на экран дисплея.

```
void output(int **z,int m, int n)
{
int i,j;
printf("\n Результирующая матрица\n");
for(i=0; i<m; i++)
{
for(j=0; j<m; j++)
printf("%8d",z[i][j] ) ;
printf("\n");
}
}
```

Следует заметить, что обычная матрица представлена в памяти как линейный массив, т.е. здесь отсутствует разбиение на строки и столбцы. В связи с этим вам самим следует позаботиться о том, чтобы операции совершались с требуемыми элементами, т.е. индексы элементов должны соответствующим образом вычисляться. Так, например, если вы ввели матрицу размером 4x4, то элемент a[3,3] будет расположен на десятой позиции в массиве, считая с нуля.

Для того, чтобы начать выполнение работы составьте главную функцию main(), в которую внесите описанные выше функции и функции обработки матриц в соответствии с вашим заданием. В начало программы нужно поместить директивы и декларации.

Пример.

```
//Примерный вид программы
#include(stdio.h)
#include(conio.h)
#include(alloc.h)
int **input_mat(int,int);
//Объявление функций обработки, использующих функцию input
int fproi(int**,int**,int,int);
.....
void ouput( int**,int,int);

void main(void)
{
int m,n;
int **p, **q;
clrscr();
puts(" Введите размер исходной матрицы");
printf("число строк=");
scanf("%d",&m);
printf("число столбцов=");
scanf("%d",&n);
p= input_mat(m,n);
output_mat(p,m,n);
q= input_mat(m,n);
output_mat(p,m,n);
fproi(p,q,m,n);
.....
}
```

При составлении программы воспользуйтесь разработками предыдущей лабораторной работы № 5.

Варианты заданий

1. T,U,S
2. MQB
3. S,M,T
4. B,U,C
5. Q,S,N
6. C,Q,T
7. U,T,N
8. S,N,U
9. C,N,T
10. M,N,Q
11. C,N,T
12. B,N,M
13. C,S,T
14. Q,C,B
15. Сравнить суммы элементов главных диагоналей.
16. Сравнить суммы элементов двух столбцов.
17. Сравнить суммы элементов двух строк.
18. Проверить: является ли матрица магическим квадратом?

В вариантах заданий использованы следующие обозначения:
T– транспонирование матрицы.
C– вычисление следа матрицы.
U– умножение матрицы на число.
M– перемножение матриц.
Q– возведение матрицы в квадрат.
S– сложение матриц
B– вычитание матриц.
N– вычисление нормы матрицы.

Содержание отчета

1. Постановка задачи.
2. Схемы алгоритмов функций.
3. Распечатки текстов программы с комментариями.

Контрольные вопросы

1. В чем состоит отличие объявление функции от ее определения?
2. Как объявить функцию с переменным числом параметров?
3. Какие типы объектов могут быть использованы в качестве формальных параметров?
4. Что такое локальные объекты? Какова их область видимости и «время жизни»?
5. В чем состоит отличие автоматических переменных от статических?
6. Объясните механизм передачи параметров по значению, по указателю, по ссылке.
7. Какой массив называется динамическим?

Лабораторная работа № 7

Организация работы с файлами

Цель работы: Овладеть навыками работы с потоками данных, находящихся на внешних устройствах.

Организация работы с файлами

Язык Си, кроме стандартного ввода данных с клавиатуры и вывода результатов на экран, предоставляет также возможность обмена данными с файлами, находящимися на диске. В Си не предусмотрены никакие предопределенные структуры файлов (такие как файлы последовательного или прямого доступа); все файлы рассматриваются как последовательные, *потоки битов*.

Для файла определен *маркер* (указатель чтения/ записи). Он определяет текущую позицию, к которой осуществляется доступ. С началом работы любой программы автоматически открываются некоторые стандартные потоки, например, стандартный ввод (его имя ***–stdin***) и стандартный вывод (его имя ***–stdout***). По умолчанию они связаны с клавиатурой и экраном соответственно. Поэтому в тех функциях ввода/вывода, которые использовались до сих пор, не указывалось, из какого потока берутся или куда помещаются данные: это известно по умолчанию.

Однако, в своей программе возможно открыть и другие потоки, связывать их либо с файлами на диске, либо с физическими устройствами (например, с принтером), записывать в них или считывать из них информацию. Для этого служат функции ввода/вывода *верхнего уровня*. Доступ к потоку осуществляется с помощью *указателя*. Указатель на файл описывается следующим образом:

```
FILE *fp;
```

Тип FILE – это структура, определенная в < *stdio*> с помощью средства *typedef* и содержащая некоторую информацию о файле: например, флаги состояния файла, размер буфера, указатель на буфер и др. Описанный указатель можно связать с конкретным файлом в момент открытия данного файла. Это осуществляется с помощью функции

fopen("путь к файлу","тип доступа"),

которая возвращает указатель на файл или NULL в случае ошибки.

Например, в результате выполнения оператора

fp=fopen("ex1.txt","w");

файл *ex1.txt* будет открыт для записи, а в программе на этот файл можно сослаться с помощью указателя *fp* (т.е. функция *fopen()* берет внешнее представление файла – его физическое имя – и ставит ему в соответствие внутреннее логическое имя, которое далее будет использоваться в программе). В качестве типа доступа могут быть указаны следующие параметры:

"w" –существующий файл открывается для записи, если файл не существовал, он создается заново, если существовал, то старое содержимое стирается.

"r" – файл открывается для чтения с начала.

"a" – файл открывается для дозаписи в конец файла.

(О других типах доступа можно прочитать в пособии по языку Си.[1].)

По окончании работы с файлом он должен быть закрыт. Для этого используется функция

fclose(указатель_файла).

Для чтения /записи данных в файл имеются функции, аналогичные уже известным функциям ввода/вывода

fprintf(), fscanf(), fputs(), fgets(), getc(), putc(), fgetc(), fputc().

Функции *getc()/ fgetc(), putc()/fputc()* по своим действиям идентичны, отличие состоит только в том, что *getc()* и *putc()* реализованы как макроопределения , а *fgetc()* и *fputc()* – как настоящие функции.

Прототипы всех файловых функций, а также необходимые константы находятся в файле *<stdio.h>*.

Чтобы продемонстрировать работу с этими функциями, рассмотрим простой пример.

Пример 1.

```
# include<stdio.h>
void main()
{
 int n;
 char str[50],str1[50],ch;
 FILE *fp;
```

```
// Заполняем файл
fp=fopen("ex.txt","w");
puts("Введите целое число") ;
scanf("%d",&n);
fprintf(fp, "%d\n" , n);
puts("Введите символ") ;
ch= getchar();
putc ( ch,fp);
puts("Введите строку") ;
gets(str);
fputs ( str, fp);
fclose ( fp);
// Читаем из файла
if(( fp = fopen ("ex.txt","r")) != NULL)
{
fscanf (fp, "%d" ,&n);
printf( "n= %d\n" , n);
ch=getc ( fp ) ; putchar (ch );
fgets ( str1, 50 , fp); puts(str1 );
fclose ( fp );
}
else printf ( "\n Нельзя открыть файл для чтения !");
}
```

Второй параметр функции *fgets()* – количество N считываемых символов, включая '\0'. Эта функция прекращает работу после чтения N-1 символа, либо после чтения '\0'. Функция *fgets()* возвращает либо адрес прочитанной строки, либо NULL по исчерпании файла или в случае ошибки.

В случае исчерпания файла или ошибки функция *getc()* возвращает EOF (End of FILE) – конец файла.

Функция *fputs ()* возвращает код последнего прочитанного символа, если все в порядке, и EOF в случае ошибки. Эта функция не переводит строку автоматически.

Все приведенные выше функции обрабатывали файл последовательно символ за символом. Язык Си предоставляет возможность работать с файлом как с массивом: непосредственно достигать любого определенного байта. Для позиционирования файла служит функция.

fseek(указатель на файл, смещение от начальной точки, начальная точка).

Второй аргумент имеет тип *long*. Его значение может быть больше и меньше нуля. Он показывает, как далеко (в байтах) следует продвинуться от начальной точки. Третий аргумент является кодом, определяющим положение начальной точки в файле. Установлены следующие значения кода: 0– начало файла, 1– текущая позиция, 2– конец файла. В случае успеха функция *fseek()* возвращает 0, а если была ошибка, то возвращается –1.

Так как язык Си связан с операционной системой UNIX, то в системе Borland C++ создана вторая система ввода/вывода. Эта система соответствует стандарту UNIX. Прототипы функций находятся в файле IO.H. Этими функциями являются

read() – читает буфер данных;
write()– пишет в буфер данных;
open()– открывает файл;
close()– закрывает файл;
lseek()– поиск определенного байта в файле;
unlink()– уничтожает файл.

Пример 2. Написать функцию, которая читает любое число байтов из любого места файла.

```
int get (int fd, long pos, char *buf, int n);
{
 if ( lseek(fd, pos, 0)>=0) return read (fd, duf, n);
 else return –1;
}
```

Задание на выполнение работы

1. Создать файл в директории группы.
2. Введите программу примера 1. Выполните действия с заполнением и считыванием символов и строк.
3. Составить программу обработки текстового файла в соответствии с заданием:

3.1. В текстовом файле подсчитать количество строк, которые оканчиваются заданной буквой (задается преподавателем).

3.2. Найти максимальную длину строки в текстовом файле и распечатать все строки файла, имеющие такую длину.

3.3. Подсчитать количество пустых строк в файле.
3.4. Подсчитать в текстовом файле количество строк, начинающихся с определенной буквы латинского алфавита.

Содержание отчета

1. Краткие теоретические сведения о методах обработки потоков данных.
2. Программа обработки, распечатки файлов с данными.
3. Комментарии к программе и полученным результатам

Контрольные вопросы

1. Что такое поток данных?
2. В чем состоит различие ввода данных в стандартном потоке и с внешнего устройства?
3. Напишите команды открытия файла для чтения и записи, закрытия файла.
4. В чем состоит различие команд *fputs()* и *fgets()*?

Лабораторная работа № 8

Обработка списков

Цель работы: освоение методов обработки линейных и дважды связанных списков, приобретение навыков составления и отладки соответствующих программ

Методические указания по обработке списков

Рассмотрим в качестве примера организацию систем учета вкладов в отделении сбербанка. Будем считать, что в простейшем случае информация о вкладе содержит фамилию вкладчика и сумму вклада в рублях. Запишем эту информацию в виде структуры:

```
struct vklad
{
char family[12];
float sum;
}
```

Пусть стоит задача разместить вклады в алфавитном порядке вкладчиков. Операции, которые надо произвести в программе, следующие: во-первых, нужно определить: имеется ли вкладчик с данной фамилией (НАЙТИ), во-вторых, включить новый вклад (ВКЛЮЧИТЬ), в-третьих, вычеркнуть вклад из системы (ИСКЛЮЧИТЬ).

Для обработки данных обо всех вкладах можно использовать массив вкладов:

```
struct vklad array [1000],
```

тогда поиск вклада может быть выполнен быстро, например, при числе вкладов К при использовании метода деления пополам максимальное время поиска примерно пропорционально целой части двоичного логарифма К.

Однако, операция *ВКЛЮЧИТЬ* в общем случае требует сдвига части массива. Удаление вклада приводит либо к незаполненным «окнам», когда информация стирается (например, поле family элемента массива array заполняется пробелами), либо опять требует сдвига части массива. Сдвиг массива – это нежелательное действие, которое занимает время, пропорциональное длине массива К.

Разрешить возникшую проблему выполнения операции *ВКЛЮЧИТЬ* и

ИСКЛЮЧИТЬ можно в результате отказа от статического размещения элементов массива (вкладов) и организации динамического списка. Список состоит из элементов, каждый из которых в общем случае является структурой, в которой выделены содержательная и вспомогательная части. Вспомогательная часть используется для организации связей между элементами списка и содержит одно или несколько полей ссылочного типа.

В программе учета вкладов, использующей динамические списки, содержательная часть элементов остается такой же, как указано выше, и к ней добавляется в простейшем случае одно поле для указания следующего элемента списка (sled). Переменные – указатели sled должны быть переменными ссылочного типа. Элементы списка можно представить в виде следующего описания:

```
struct vklad
{
char family[12];
float sum;
struct vklad sled;
}
```

Тогда информацию, например о четырех вкладах, можно представить в виде списка. Переменная top типа struct vklad* указывает на первый элемент списка, а поле sled в последнем элементе имеет значение NULL.

Сформировать такой список можно с помощью программы включающей следующий фрагмент:

```
struct vklad *top;
struct vklad *p;
……..
top=NULL;
for(i=0; i<4; i++)
{
 p = ( struct vklad *) malloc((sizeof (struct vklad )));
 p--> sled=top;
 printf("Введите фамилию вкладчика:") ;
 gets(p-->family);
 printf("Введите сумму вклада:");
 scanf("%f",p->sum);
 top=p;
}
```

Рассмотрим поиск элемента списка с заданным значением одного из полей. Пусть, например, необходимо увеличить сумму вклада кладчика А. Для этого рассмотрим еще одну переменную типа strukt vklad *pt, которая будет перемещаться по списку до обнаружения нужной фамилии:

```
pt=top;
while(pt–> family !=A)
pt=pt–>sled;
```

Этот фрагмент можно пояснить так. Сначала pt указывает на первый элемент списка. До тех пор пока фамилия вкладчика, на которую указывает переменная pt, не будет совпадать с заданной фамилией А, нужно передвигать pt на переменную, задаваемую полем структуры, на которую в данный момент выполнения указывает pt.

Приведенный фрагмент будет решать задачу ПОИСК только в том случае, когда можно гарантировать, что человек с фамилией А имеет в системе учета некоторый вклад.

Если это не так, то, «подойдя» к последнему элементу списка, переменная pt приобретает значение NULL, и на следующем шаге выполнения цикла обращение к полю pt–> family вызовет аварийное окончание программы, так как требуемого элемента просто не существует.

Опознать конец списка можно попытаться так:

```
pt=top;
while( pt!=0 && (pt–> family !=A))
pt=pt–>sled;
```

Это решение в общем случае не устранит аварийное окончание по той же самой причине : в конце списка pt==NULL и элемента pt с полем family, который требуется для выполнения операции сравнения с А в программе учета не создано.

Два варианта правильного решения приведены ниже:

1)
```
pt=top;
while(pt–>sled !=NULL) && (pt–> family !=A))
pt=pt–>sled;
if ( pt–> family != A)
pt=NULL;
```

2)
```
pt=top;
int b=1;
```

```
while(pt !=NULL && b)
b=0;
else
pt=pt–>sled;
```

В обоих случаях после выполнения фрагмента справедливо положение: если человек с фамилией A является вкладчиком, то pt указывает на его вклад, иначе pt==NULL.

Время поиска элемента с заданным полем пропорционально длине списка, так как при поиске просматривается последовательно весь список.

Следующая типовая задача состоит в том, чтобы вставить новый элемент в список. Для нашего примера это соответствует появлению нового вклада. Вставка возможна в начало списка (перед элементом, на который указывает переменная top) или после любого элемента, например, после элемента, на который указывает pt.

```
struct vklad *new ;
new=( struct vklad *) malloc(sizeof (struct vklad ));
new–>sled=pt –> sled ;
pt–>sled=new;
```

Существуют два вида списков: связанный и кольцевой. Связной список показан на рис.8.1. Это простой *однонаправленный* список, в котором каждый

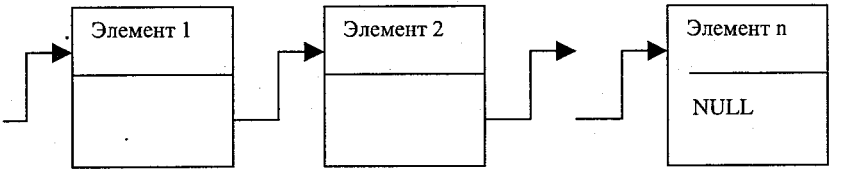

Рис. 8.1

элемент (кроме последнего) имеет ссылку на следующий элемент и поле информации. Можно организовать также *кольцевой* список (в нем последний элемент будет содержать ссылку на первый) или *двунаправленный* список (рис.8.2), когда каждый элемент, кроме первого и последнего, имеет две ссылки : на предыдущий элемент и следующий элемент и т.д.

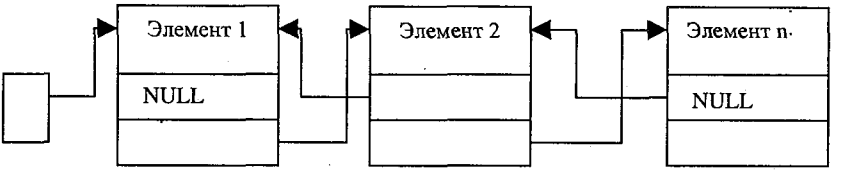

Рис. 8.2

Кроме того, можно помещать в начале списка дополнительный элемент, называемый заголовком списка, который может использоваться для хранения информации обо всем списке. В частности, он может содержать счетчик числа элементов списка.

Элемент дважды связанного списка содержит два поля указателей, один из которых указывает на следующий элемент, а другой – на предыдущий. Такая организация позволяет перемещаться списку в двух направлениях: влево и вправо. Циклический список представляет собой «кольцо» элементов и является простой модификацией рассмотренного выше линейного списка: последний элемент в поле связи вместо значения NULL содержит ссылку на первый элемент списка.

Рассмотрим пример оформления функции, формирующей дважды связанный линейный список. В «содержательной» части элементов списка записан один символ. Формирование списка заканчивается при вводе символа '*'.

```
struct element
{
char inf;
struct element *lev;
struct element * prav;
}
.....

void form (struct element * perv)
{
struct element *tek;
char ch;
perv =NULL;
do
```

```
{
tek=( struct element *) malloc(sizeof (struct element )) ;
tek –> prav = perv;
tek –> lev = NULL;
scanf("%c" , ch);
tek –> inf = ch;
perv = tek;
if (tek –> prav !=NULL)
tek –> prav –>lev = tek;
}
while(ch=='*');
}
```

Ниже приведен заголовок функции, осуществляющий удаление элемента, содержательное поле которого совпадает с заданным значением. Параметрами функции является ссылочная переменная-указатель головы списка и заданный символ.

```
void udalen ( struct element* top, char ch );
```

Задание к работе

Составить программу обработки списка в соответствии с номером задания. Вид списка определяется номером строки табл.8.1, а вид обработки – номером столбца, в котором стоит номер задания по табл.8.2. Например, для задания 12 следует рассматривать линейный дважды связанный список и произвести проверку: входит ли в список заданный элемент с заданным значением информационного поля?

В программе предусмотреть функцию формирования списка, функцию его вывода и функцию обработки. Тело программы представляет собой последовательность четырех обращений к функциям: ввод списка, его контрольный вывод, обработку, вывод результата.

Таблица 8.1

ВИД СПИСКА

Вид списка	1	2	3	4	5	6	7	8	9	10
Линейный			20		25	23	4	21	24	19
Линейный дважды связанный		18	6	8	10	14	12	1	5	9
Линейный циклический	16	4	7	17	22	11	13	2	15	3

Таблица 8.2

ВИД ОБРАБОТКИ СПИСКА

Номер	Вид обработки
1	Подсчитать число элементов списка
2	Добавить новый элемент. Элемент задан ссылочной переменной
3	Добавить новый элемент после заданного. Элемент задан значением информационного поля.
4	Удалить заданный элемент из списка. Элемент задан ссылочной переменной.
5	Удалить заданный элемент. Элемент задан значением поля; удаляется первый элемент
6	Удалить заданный элемент. Элемент задан значением поля; удаляются все такие элементы.
7	Проверить: входит ли заданный элемент в список?
8	Подсчитать: сколько имеется элементов с заданным содержимым одного из полей
9	Объединить два списка– второй добавить в хвост первого (циклические списки разрываются в произвольном месте)
10	Найти элемент с заданным значением информационного поля (из функции обработки возвращается значение ссылочной переменной, указывающей на первый элемент).

Для сохранения списка и последующего вывода его на печать в программе предусмотреть файл для чтения и записи. Файл должен содержать все необходимые поля, представленные в виде таблицы.

Содержание отчета

1. Цель работы и индивидуальное задание.
2. Алгоритм решения задачи с необходимыми комментариями.
3. Рисунки, поясняющие процессы обработки списка.
4. Разработанную программу.
5. Сформированный текстовый файл.

Контрольные вопросы

1. Какой список называется линейным?
2. Какой список называется кольцевым?
3. Как проверить: входит ли заданный элемент в список?
4. Как найти список с заданны значением информационного поля?

Лабораторная работа № 9

Вывод графиков функций

Цель работы: Изучить методы построения графиков функций в графическом редакторе языка Си. Составить программу для расчета и вывода в графическом режиме заданной функции. Результаты расчета выводятся в графической форме на экран и принтер.

Исходные данные: Математическая функция, диапазоны изменения параметров функции.

Пример построения графика

Напишем программу, которая выводит на экран точечный график функции $y=0,5x^2+4x-3$. Диапазон изменения аргумента: от –15 до 5; шаг аргумента –0,1.

График вывести на фоне оцифрованной координатной сетки. Начало координат должно находится в центре экрана. Предусмотреть вывод графика в виде точечной кривой и сплошной линии какого-либо цвета. Оси координат вывести в виде сплошной линии цвета фона. Линии координатной сетки вывести тонкими точечными линиями.

Программу целесообразно составить из двух функций: функции grid(), выводящей на экран оцифрованную координатную сетку, и функции grafik(), строящей график функции.

```
void grid()
{
int x0,y0; //начало координат
int dx,dy; //шаг координатной сетки ( в пикселах)
int h,w; //высота и ширина области вывода
int x,y;
float lx,ly; // метки линий сетки по X и Y
float dlx,dly; // шаг меток линий сетки по X и Y
char st[8]; // изображение метки линии сетки
x0=50; y0=400; // начало осей координат
dx=40; dy=40; //шаг приращения по осям
dlx=0.5;
dly=1;
h=300;
w=400;
```

```
lx=0;
ly=0;
line (x0,y0,xo,y0–h); //ось X
line (x0,y0,xo+w,y0); // ось Y
/*засечки, сетка и оцифровка по оси X*/
x=x0;
do
{
//засечка
setlinestyle(SOLID_LINE,0,1); //вид линии
line (x,y0–3,x,y0+3);
//оцифровка
sprintf(st,"%2.1f",lx);
outtextxy(x–8,y0+5,st);
lx+=dlx;
//линия сетки
setlinestyle(DOTTED_LINE,0,1);
line (x,y0–3,x,y0–h);
x+=dx;
}
while(x<x0+w);
/*засечки, сетка и оцифровка по оси Y*/
y=y0;
do
{
//засечка
setlinestyle(SOLID_LINE,0,1);
line (x0–3,y,x0+3,y);
//оцифровка
sprintf(st,"%2.1f",ly);
outtextxy(x0–40,y,st);
ly+=dly;
//линия сетки
setlinestyle(DOTTED_LINE,0,1);
line (x0+3,y,x0+w,y);
setlinestyle(SOLID_LINE,0,1);
y–=dy;
}
```

```
while (y>y0–h);
} // конец программы grid().
```
Построим точечный график для функции $y=0,5x^2+4x–3$.
```
#include <graphics.h>
 #include<math.h>
    #include<stdlib.h>
    #include <conio.h>
    #include<stdio.h>
    #include<zaggraf.c>
    void grafik()
    {
    float x,dx;
    float x1,x2;
    float y;
    int mx,my;
    int x0,y0;
    int px,py;
    x0=320;y0=240;
    mx=20;my=20;
    line(10,y0,630,y0);//ось x , строится если нет сетки
    line(x0,10,x0,470);//ось y, строится если нет сетки
    x1=–25;
    x2=5;
    dx=0.1;
    x=x1;
    moveto(x,0);
    while(x<x2)
    {
 y=0.5*x*x+x*4–3;
 px=x0+x*mx;
    py=y0–y*my;
 putpixel(px,py,WHITE);//построение графика по точкам
    setcolor(WHITE);
    //lineto(px,py); в случае сплошного графика
    x+=dx;
    }
    }
```
Запишем теперь главную функцию.

```
void main(void)
{
zaggraf(); // загрузка графики
grafik(); // ввод функции графика
grid(); // ввод функции сетки
getch();
closegraph();
}
```

Задание на выполнение работы

Разработать программу построения графика одной из функций. Варианты заданий приведены в табл.9.1.

Таблица 9.1

ВИДЫ ФУНКЦИЙ

№ варианта	Функция	Диапазон изменения аргумента
1	$y= 1.3x^2–1.8+\log(x)$	–1.2,1.2
2	Окружность x=0.5+2cos(t); y=0.2+2sin (t)	$0, 360^0$
3	Степенная функция $y=x^3–2x^2+x$	–1,3
4	Эллипс x=3cos(t); y=15sin(t)	0, 360
5	Показательная функция $y=\exp(x^2)$	–1.3, 1.3
6	Кардиоида x=4cos(t) (1+cos(t)) y=4sin(t)(1+cos(t))	0, 20
7	Дробно-рациональная функция y= (1.5x +3) /(x–2)	–4.2, 1.9
8	Декартов лист $x=3at /(1 + t^3)$ $y=3at^2 /(1 + t^3)$	 –0.5, 10 A=2
9	Функция синус y=2.5sin(x) + 0.5	–20, 20
10	Циссоида $x=5t^2 /(1 + t^2)$ $y=5t^2 /(1 + t^2)$ t=tg(f)	–0/4, 0/4
11	Тригонометрическая функция $y=\cos(x^2)$	–20, 20
12	Строфоида $x=4(t^2–1) /(t^2+1)$ $y=4t(t^2–1) /(t^2+1)$ t=tg(f)	–0/2.5, 0/2.5
13	Тригонометрическая функция y=tg(x) – 2x	–0/2.5, 0/2.5
14	Астроида $x=3.5\cos^3(t)$, $y=3.5\sin^3(t)$	0, 20
15	Арксинус y=arcsin (0.5x)	x –2, 2
16	Эпициклоида x=(a+b)cos(t) – acos ((a+b)t/a), y=(a+b)sin(t) – asin ((a+b)t/a)	a=6, b=9 t 0, 20

17	Логарифм y=ln(x+2)	–1.5, 5
18	Гопоциклоида x=2acos(f) + acos(2f), y=2asin(f) + acos(2f)	–90, +90 a=1.5
19	Арктангенс y=3 arctg(x)	–5, 5
20	Эвольвента окружности x=a cos(f) + af sin(f) y=a sin(f) – acos(f)	f=[–90,90] a=1.5
21	Дробно- рац. нелинейная y=ax+c/ bx+cx^2	0.18, 3 a=1, b=2, c= -0.5
22	Лемниската x=rcos(f) y=rsin(f) r=asqrt(2cos(2f))	0,10
23	Локон Аньези y= a^3/x^2 + a^2	–5, 5 a=2
24	Архимедова спираль x=rcos(f) y=rsin(f) ,r=af	–60, 60 a=1.5
25	Трохоида (Удлиненный цикл) x=a(1+b sin(f)) y=a(1–b cos(f))	–20, 40 a=1.5, b=1.3
26	Гиперболическая спираль x=(a cos(f))/f y=(a sin(f))/f	0.1, 10 a=3
27	Удлиненная эпициклоида x=5cos(f) – 2cos(5f) y=5sin(f) – 2sin(5f)	0, 10
28	Логарифмическая спираль x=rcos(f), y=rsin(f) r=aexp(bf)	f =[0, 40] a=1.3, b=0.5
29	Удлиненная гипоциклоида x=4cos(f) + 2cos(4f) y=4sin(f) – 2sin(4f)	f 0, 20
30	Конхоида Никомеда x=a+bcos(f) y=atg(f) + bsin(f)	–1.5, 1.5 a=1, b=2
31	Улитка Паскаля x=2cos^2(t) + 3cos(t), y=2cos(t)sin(t) + 3sin(t)	0, 2*Pi

Содержание отчета

1. Краткие теоретические сведения о графических методах обработки данных.
2. Программа построения графика заданной функции.
3. Комментарии к программе и полученным результатам.
4. Распечатку графика функции.

СПИСОК РЕКОМЕНДУЕМЫХ ИСТОЧНИКОВ

1. *Керниган Б., Ритчи Д., Фьюэр А.* Язык программирования Си. – М.: Финансы и статистика , 2000.
2. *Березин Б. И., Березин С. Б.* Начальный курс С и С++. – М.: ДИАЛОГ – МИФИ, 1999.
3. *Культин Н.Б.* С/С++ в задачах и примерах – СПб.: БХВ– Петербург, 2001.
4. *Крячков А.В., Сухинина И.В., Томшин В.К.* Программирование на С и С++. Практикум: Учеб. пособие для вузов / Под ред. В.К.Томшина – 2-е изд., исправ.– М.: Горячая линия – Телеком, 2000.

СОДЕРЖАНИЕ

Учебное издание

ПРАКТИКУМ

«Программирование на языке Си»

Анатолий Дмитриевич Шишкин
Елена Анатольевна Чернецова

Редактор И.Г. Максимова

ЛР № 020309 от 30.12.96.

Подписано в печать 20.03.03. Формат 60x90 1/16. Гарнитура Times New Roman.
Бумага офсетная. Печать офсетная. Печ.л. 3,1. Уч.-изд.л. 3,5. Тираж 200 экз. Заказ № 13
РГГМУ, 195196, Санкт-Петербург, Малоохтинский пр., 98.
ЗАО «Лека», 195112, Санкт-Петербург, Малоохтинский пр., 68.