Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра электронных вычислительных машин

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Лабораторный практикум
для студентов специальности
1-40 02 01 «Вычислительные машины, системы и сети»
всех форм обучения

В 2-х частях

Часть 2

Минск БГУИР 2010

УДК 004.421(075.8)
ББК 32.973.26-018.2я73
О-75

Составители:

Ю. А. Луцик, А. М. Ковальчук, И. В. Лукьянова, А. В. Бушкевич

Рецензент:

заведующий кафедрой экономической информатики
учреждения образования «Белорусский государственный университет
информатики и радиоэлектроники»,
кандидат технических наук, доцент В. Н. Комличенко

Основы алгоритмизации и программирования : лаб. практикум для О-75 студ. спец. 1-40 02 01 «Вычислительные машины, системы и сети» всех форм обуч. В 2 ч. Ч. 2 / сост. Ю. А. Луцик [и др.]. – Минск : БГУИР, 2010. – 36 с. : ил.

ISBN 978-985-488-476-9 (ч. 2)

Содержит описание и порядок выполнения семи лабораторных работ по темам курса. В каждой работе даны краткие теоретические сведения, в конце приведены варианты заданий.

УДК 004.421(075.8)
ББК 32.973.26-018.2я73

Часть 1 издана в БГУИР в 2007 г.

ISBN 978-985-488-476-9 (ч. 2)
ISBN 978-985-488-477-6

Лабораторная работа №1

Структуры

Цель работы: разработать алгоритмы и написать программы для работы со структурами.

Теоретические сведения

В том случае, когда под одним именем необходимо собрать различные типы данных, используются структуры [3].

Общая форма объявления структуры является основной схемой, описывающей, как собирается структура:

```
struct book            //  объявление структуры
{
   char tit[10];
   char aut[20];
   float value;
};
```

Ключевое слово struct определяет, что все, что следует за ним в фигурных скобках, является структурой, а идентификатор рядом с ключевым словом определяет имя типа структуры. Структуры могут копироваться, их можно передавать функциям в качестве аргументов, а функции могут возвращать их в качестве результатов. Доступ к полям структуры осуществляется через структурную переменную, например:

```
struct book libry;      // описание переменной libry, имеющей тип book
```

Под переменную libry отводится память (рис. 1).

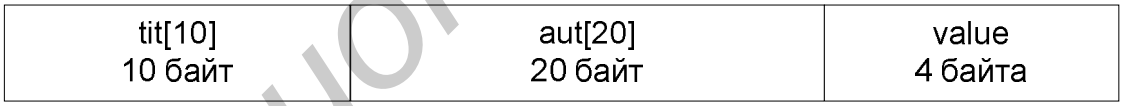

Рис. 1. Выделение памяти под структуру

Массив структур объявляется следующим образом:

```
struct book libry[2];
```

Под объявленный массив будет отведена память (рис. 2).

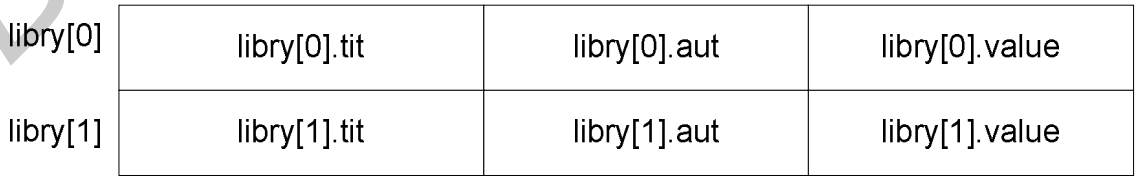

Рис. 2. Выделение памяти под массив структур

Для обращения к отдельным элементам структуры используется оператор точка «.». Например:

```
struct book                 // объявление структуры
{
        char tit[40];        //  название книги
        char aut[20];        //  фамилия автора
        float value;         //  цена книги
};
struct book libry[4];       // массив структур
for(int i=0; i < 4; i++)
{       printf("\n Введите название книги");
        gets(libry[i].tit);
        printf("\n Введите ФИО автора");
        gets(libry[i].aut);
        printf("\n Введите цену книги");
        scanf("%f",&(libry[i].value));
        fflush(stdin);
}
```

Можно объявить указатель на структуру следующим образом:

```
struct book *pt;
```

Указатель необходимо проинициализировать, т. е. присвоить ему некоторый адрес, например, адрес массива n структур типа book:

```
pt= (book *)malloc(n*sizeof(book));
```

В этом случае обращение к элементам структуры осуществляется либо с помощью операций «*» и «.» т. е. (*pt).tit; (*pt).aut; *(pt).value, либо, что более просто, операции «->», т. е. pt->tit; pt->aut; pt->value;

Для объявления массива структур с помощью указателя на структуру можно выполнить следующие действия:

```
struct book                 // объявление структуры
{
        char tit[40];        //  название книги
        char aut[20];        //  фамилия автора
        float value;         //  цена книги
};
int n;                      // количество структур
struct book *libry;         // указатель на структуру
printf("\nВведите количество структур");
scanf("%d",n);              // ввод количества структур
libry=(book*)calloc(n,sizeof(book));   // выделение памяти под n структур
for(int i=0; i < n; i++)
{
        printf("\nВведите название книги");
        gets(libry[i]->tit);
        printf("Введите ФИО автора");
        gets(libry[i]->aut);
```

```
        printf("\nВведите цену книги");
        scanf("%f",libry[i]->value);
        fflush(stdin);
}
```

Структуры могут включать в себя другие структуры, т. е. можно использовать вложенные структуры, например:

```
struct date           // объявление структуры типа date
{
  int day,       // день рождения
      month,    // месяц рождения
      year;      // год рождения
};
struct person       // объявление структуры типа person
{
   char fam[30],          // фамилия
 im[20],             // имя
 otch[40];          // отчество
   int   weight,             // вес
height;             // рост
struct date birthday;      // вложенная структура
};
```

Пример. Среди абитуриентов, сдавших вступительные экзамены в институт, определить количество абитуриентов, проживающих в городе Минске и сдавших экзамены со средним баллом не ниже 8, распечатать их фамилии в алфавитном порядке.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
struct str                  // шаблон структуры
{
      char *gr;             // город
      char fm[15];          // фамилия
      int oc[3];            // массив оценок
};
void main(void)
{
      struct str *s;              // указатель на массив структур
      int n;                      // количество абитуриентов
      int i,j,k=0;
      char c;
      clrscr();
      printf("\nВведите количество абитуриентов");
      scanf("%d",&n);
      s=(str *)calloc(n,sizeof(str));
```

```
    if(s==NULL)
    {
        printf("\nПамять под массив структур не выделена");
        return;
    }
    for(i=0;i<n;i++)          // цикл ввода структур в массив
    {
        printf("\nВведите информацию о %3d абитуриенте",i);
        printf("\n введите город :");
        fflush(stdin);
        s[i].gr=(char *)malloc(10);  //отводим память для ввода города
        gets(s[i].gr);               //ввод города
        if (!*s[i].gr) break;        // выход по пустой строке
        printf("\n введите фамилию :");
         gets(s[i].fm);               //ввод фамилии
         printf("\n введите оценки (ф  м  л) :");
        scanf("%d%d%d",&s[i].oc[0],&s[i].oc[1],&s[i].oc[2]);
    }
    i--;          // последняя структура  s[i] – пустая или i=n
    printf("\nВведенная информация об абитуриентах");
    printf("\n Город        Фамилия        Оценки");
    for(i=0;i<n;i++)
    {
        printf("\n");
        printf("%8s",s[i].gr);      // печать города
        printf("%8s",s[i].fm);      // печать фамилии
        printf("%5d%5d%5d", s[i].oc[0],s[i].oc[1],s[i].oc[2]);
    }
    for(c='A';c<='Я';c++)
      for(j=0;j<n;j++)
        if(!strcmp(s[j].gr,"Минск") && s[j].fm[0]==c)
        if((s[j].oc[0]+s[j].oc[1]+s[j].oc[2])/3>=8)
            k++;          // подсчет числа абитуриентов, удовлетворяющих
                          // условию задачи
  printf("\n Итого абитуриентов = %4d",k);
}
```

Порядок выполнения работы

1. Изучить краткие теоретические сведения, материал лекции по теме лабораторной работы.
2. Составить блок-схему алгоритма.
3. По разработанной блок-схеме алгоритма написать программу.
4. Отладить и выполнить программу.

Варианты заданий

1. Имеются сведения о веществах (проводник, полупроводник, изолятор): название вещества, его удельный вес и проводимость:

а) найти удельные веса и название всех полупроводников;

б) выбрать данные о проводниках и упорядочить их по убыванию удельных весов.

2. Имеются сведения о различных датах. Каждая дата – это число, месяц и год. Найти:

а) год с наименьшим номером;

б) все весенние даты;

в) самую позднюю дату.

3. Сведения об ученике состоят из его имени, фамилии и названия класса (год обучения и буква), в котором он учится:

а) выяснить, имеются ли в школе однофамильцы;

б) выяснить, имеются ли однофамильцы в каких-либо параллельных классах.

4. Сведения об ученике состоят из его имени, фамилии и названия класса (год обучения и буква), в котором он учится, и оценок, полученных учеником в последней четверти:

а) выяснить, сколько учеников школы не имеют оценок ниже четырех;

б) выдать сведения о лучших учениках школы, т. е. об учениках, не имеющих оценок ниже четырех и по сумме баллов не уступающих другим ученикам своего и параллельных классов.

5. Имеются сведения об игрушках: название игрушки (например, кукла, кубики, мяч, конструктор и т. д.), ее стоимость и возрастные группы детей, для которых игрушка предназначена (например для детей от двух до пяти лет). Получить следующие сведения:

а) названия игрушек, цена которых не превышает заданной суммы и подходит детям 5 лет;

б) цену самого дорогого конструктора.

6. Имеются сведения о книгах – это фамилия автора, название и год издания:

а) найти названия книг данного автора, изданных с n-го года;

б) определить, имеется ли книга с указанным названием. Если да, то сообщить фамилию автора и год издания. Если таких книг несколько, то сообщить имеющиеся сведения обо всех этих книгах.

7. Имеются сведения о кубиках: размер каждого кубика (длина ребра в сантиметрах), его цвет (красный, желтый, зеленый и синий) и материал (дерево, металл, картон). Найти:

а) количество кубиков каждого из перечисленных цветов и их суммарный объем;

б) количество деревянных кубиков с ребром n сантиметров и количество металлических кубиков с ребром большим m сантиметров.

8. Имеются сведения об экспортируемых товарах: наименование товара; страна, импортирующая товар; и объем поставляемой партии в штуках. Найти страны, в которые экспортируется данный товар и общий объем его экспорта.

9. Имеются сведения о студентах: ФИО, домашний адрес, дата рождения. Сформировать массив, в который записать в алфавитном порядке студентов, которым исполнилось n лет. Ввод, поиск, вывод выполнять в разных функциях.

10. Обработать информацию о фирмах городов. Информация следующая:

– название фирмы (не более тридцати знаков);

– величина налогообложения (не более 1 млн р. – в виде строки);

– дата (месяц – в виде строки) последнего срока внесения налога;

– дата его фактического внесения (строка).

В одной функции внести названия фирм, в другой – величину налога, в третьей – предельную дату внесения налога и дату, когда налог погашен (если не внесен, то вводится нуль). В головном модуле для заданной даты (месяц) вывести в алфавитном порядке пять фирм, имеющих максимальную задолженность. Глобальные переменные, системные функции, кроме функций ввода-вывода, не использовать. Исходный массив сохранить, новых массивов структур не создавать. Можно объявлять и вводить другую необходимую информацию.

Л а б о р а т о р н а я р а б о т а №2

Объединения. Поля бит

Цель работы: научиться использовать объединения и поля бит при разработке и написании программ.

Теоретические сведения

Объединение – это переменная, которая позволяет в разные моменты времени хранить значения различных типов в одном и том же месте памяти. Объединение позволяет создавать массив из элементов одинакового размера, каждый из которых может содержать различные типы данных. Различаются объявление объединения и описание переменной типа объединения.

Объединение объявляется следующим образом:

```
union name          // объявление объединения name
{    int d;
     double b;
     char f;
};
union name ft;              // ft – переменная типа объединения hh
union name mas[5];      // массив объединений
union name *pu;          // указатель на переменную типа объединения hh
```

Под переменную ft компилятор выделяет достаточно памяти для размещения самой большой из описанных переменных. Вот как используется объединение:

```
ft.d=23;          // 23 записывается в ft
ft.b=2.5;         // 23 стирается, а 2.5 записывается
ft.f='h';         // 2.5 стирается, а записывается код символа h
```

Имена полей в одном объединении должны быть уникальными. Однако в разных объединениях можно использовать совпадающие имена полей. В каждый момент времени запоминается только одно значение. Если объявлен указатель на объединение, то сначала этот указатель необходимо проинициализировать pu=(name*)malloc(k*sizeof(name)); . Доступ к элементам объединения в этом случае осуществляется, как и в случае структур, например, через операцию «->», т. е. pu->d;, pu->b;, pu->f; .

Язык С допускает использование в структурах особого типа полей – так называемых битовых полей. Использование данных полей делает возможным доступ к отдельным битам более крупных объектов, например байт или слов. Поля бит удобно использовать тогда, когда для хранения информации в структуре данных достаточно нескольких бит.

Общий синтаксис описания битового поля:

тип [имя]: ширина; .

В языке С для определения битового поля разрешено использовать любой тип, интерпретируемый как целый: char, short, int, long (signed или unsigned), перечисления. В полях типа signed крайний левый бит является знаковым. Каждому полю выделяется точно столько бит, сколько указано в поле «ширина». Ссылка на битовое поле выполняется по имени, указанному в поле «имя». Если имя в данном поле не указано, то запрошенное количество бит будет выделено, но доступ к ним будет невозможен. Поля бит размещаются в машинном слове справа налево (т. е. в направлении от младших разрядов к старшим) в очередности их объявления. Если объявлена следующая структура:

```
struct имя_структуры
{
      тип [имя1]: ширина;
      тип [имя2]: ширина;
      . . . . . .
      тип [имяN]: ширина;
};
```

то размещение полей в памяти можно представить следующим образом (рис. 3).

старшие – РАЗРЯДЫ МАШИННЫХ СЛОВ – младшие
имяN имя2 имя1

Рис. 3. Поля бит в структуре

Пример. Используя поля бит и объединения, написать программу, вычисляющую остаток от деления целого числа на 2 и 4.

```
#include <stdio.h>
#include<conio.h>
int main(int)              // вычисление остатка от деления на 2, 4
{
      struct pole
      {
            union
            {
                  struct
                  {
                        unsigned i1:1;
                        int:1;
                  } st1;
                  struct
                  {
                        unsigned i1:2;
                  }st2;
            } un;
            int:14;
      };
       pole *pl;
       int k;
       scanf("%d",&k);
       pl=(pole *)&k;
       clrscr();
       printf("\n остаток от деления на 2 =%d",pl->un.st1.i1);
       printf("\n остаток от деления на 4 =%d",pl->un.st2.i1);
 }
```


Порядок выполнения работы

1. Изучить краткие теоретические сведения.
2. Составить блок-схему алгоритма.
3. По разработанной блок-схеме алгоритма написать программу.
4. Отладить и выполнить программу.

Варианты заданий

1. В головном модуле ввести структуры. Элементами структуры является вторая структура (фамилия, имя, отчество студента и номер семестра) и объединение. В объединение в зависимости от семестра записаны номера экзаменов и результаты, а список экзаменов хранится отдельно. В зависи-

мости от запроса в командной строке вывести информацию о студентах за определенный семестр.

2. В массиве структур хранится информация о детях детского сада. Элементом структуры является объединение, в котором хранится следующая информация: последнее заболевание ребенка (грипп, ангина и т. д.) и имя участкового врача. Если ребенок находился в больнице, то и номер больницы, ее адрес и фамилия лечащего врача. По запросу из командной строки вывести в алфавитном порядке фамилии детей, имеющих данное заболевание.

3. В командной строке задаются два признака. Первый признак определяет тип вводимой информации: просто структура или структура, включающая объединение и имена функций, которые надо выполнить. Функции выводят информацию (из просто структуры или из структуры, включающей объединение) в зависимости от второго признака в командной строке.

В структурах хранится информация о студентах: фамилия, имя, отчество.

В объединении структура хранит один из типов информации:

а) отец, мать, брат;

б) отец, мать, брат, сестра;

в) мать, брат, сестра.

4. В структурах хранится информация о студентах. Одним из элементов структуры является объединение, в котором в зависимости от места жительства (иногородний студент или нет) информация задается в виде:

а) Минск, ул. …, д. …, кв. …;

б) область, город …, ул. …, д. …, кв. …;

в) область …, район …, город …, ул. …, д. …, кв. …;

г) область …, район …, деревня …, дом … .

По запросу из командной строки в зависимости от вида информации вывести данные о тех или иных студентах с помощью указателей на функции.

5. Имеется массив структур с информацией о студентах. В структуре в качестве подструктуры задаются фамилия, имя, отчество студентов. Их медицинские параметры задаются в виде объединения в этой же структуре. По запросу из командной строки выдать информацию о студентах с соответствующими признаками.

Объединения включают:

а) рост, вес;

б) рост, вес, два-три других параметра.

6. В головном модуле ввести массив структур. Одним из элементов структуры является объединение. В нем хранится количество валюты, которая есть у человека. В зависимости от запроса вывести: кто самый богатый, бедный, у кого больше белорусских рублей.

7. Используя массив структур, одним из полей которого является объединение, содержащее информацию об одной из фигур, например, треугольник, окружность, трапеция, вычислить площадь этой фигуры и сохранить в другом поле массива структур. Организовать ввод информации в массив и вывод результатов вычисления.

Л а б о р а т о р н а я р а б о т а №3

Динамические структуры данных. Стек

Цель работы: изучить принципы построения динамических структур данных, организацию данных в виде стека; разработать алгоритмы и написать программы для работы со стеком.

Теоретические сведения

Список – совокупность объектов (элементов списка), размещаемых в динамической памяти, в которой каждый объект содержит информацию о местоположении связанного с ним другого объекта. В простейшем случае элемент списка представляет собой структурную переменную, содержащую указатель (указатели) на следующий элемент и любое число других полей (информационных).

Списки бывают линейными и кольцевыми, односвязными и двусвязными.

Список называется односвязным (однонаправленным), если движение от элемента к элементу списка возможно только в одном из направлений. Список имеет начальную точку такого движения, элемент этого списка включает только указатель на следующий элемент.

В случае двусвязного (двунаправленного) списка возможно движение от элемента к элементу в обоих направлениях. При этом элемент содержит два указателя: на предыдущий и последующий элементы списка.

Самыми распространенными случаями линейного односвязного списка являются стек и очередь.

Стек – упорядоченный набор элементов, в котором размещение новых элементов и удаление существующих производится только с одного его конца, называемого вершиной стека. Таким образом, стек является динамической, постоянно изменяющейся структурой (рис. 4).

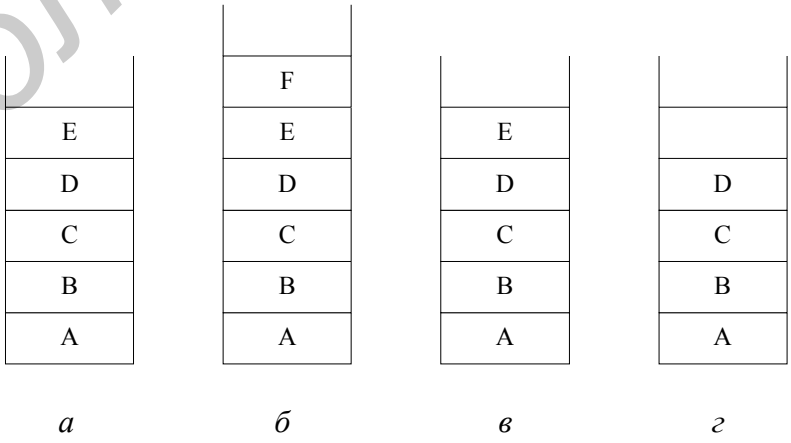

Рис. 4. Организация стека

В какую сторону растет стек, какой конец стека является его вершиной, с какого его конца будут добавляться и удалятся элементы? Элемент E (рис. 4, *а*) в данный момент является вершиной. Если в стек помещается новый элемент F, то он становится новой вершиной стека (рис. 4, *б*). Удалять из стека можно элемент, находящийся на его вершине (F на рис. 4, *б*), при этом новой вершиной становится следующий за ним элемент E (рис. 4, *в*), далее при удалении E вершиной станет следующий за ним элемент D (рис. 4, *г*).

Согласно определению в стеке есть только одно место для размещения новых элементов – его вершина. Последний размещенный элемент находится в вершине стека. Стек иногда называют списком с организацией «последний размещенный извлекается первым» (LIFO – Last In, First Out).

Пример. Рассмотрим программу, иллюстрирующую работу со стеком:

```
#include <stdio.h>
#include <alloc.h>
#include <conio.h>
#include <string.h>
struct zap
{
      char inf[50];          // информационное поле
      struct zap *l;         // адресное поле
};
void see(zap *);             // просмотреть стек
zap* butld(zap *);           // создать вершину стека
zap * add(zap *);            // добавить элемент в стек
zap * del(zap *);            // удалить элемент стека
void main(void)
{
      zap *s;
      s=NULL;
      clrscr();
       while(1)
      {
          puts("вид операции:A – создать/добавить");
          puts("                D – удалить");
          puts("                S – просмотреть");
          puts("                E – закончить");
          fflush(stdin);
          switch(getch())
          {
                case 'a': case 'A': s=add(s); break;
                case 'd': case 'D': s=del(s); break;
                case 's':  case 'S': see(s); break;
                case 'e':  case 'E': return;
```

```
                default: printf("Ошибка, повторите \n");
            }
        }
}
// функция создания и добавления элемента стека
struct zap * add(struct zap *s)
{
        struct zap *s1;
         s1=s;        // текущая вершина стека
        if((s=(struct zap *)malloc(sizeof(struct zap)))==NULL)
        {               // размещаем новый элемент на вершину стека
                puts("Нет свободной памяти");
                return 0;
        }
        puts("Введите информацию в inf");
        scanf("%s",s->inf);
        s->l=s1;       //  указатель на предыдущий элемент стека
        return s;
}
//функция просмотра элементов стека
void see(struct zap *s)
{
        struct zap *s1;
        s1=s;          // текущая вершина стека
        if(s==NULL)
        {
                puts("Стек не создан");
                return;
        }
        do
        {
                printf("%s\n",s1->inf);
                s1=s1->l;       // переход к предыдущему элементу стека
        }while(s1!=NULL);
        puts("вывод стека закончен");
}
// функция удаления последнего элемента стека
struct zap * del(struct zap *s)
{
        struct zap *s1;
        if(s==NULL)
        {
```

```
        puts("Стек пуст");
        return 0;
    }
    s1=s;        // запоминаем старый указатель на вершину стека
    s=s->l;      // передвигаем указатель на следующий элемент стека
    free(s1);    // удаляем элемент с вершины стека
    puts("последний элемент стека удален");
    return (s);  // возвращаем указатель на новую вершину стека
}
```

Порядок выполнения работы

1. Изучить краткие теоретические сведения.
2. Составить блок-схему алгоритма.
3. По разработанной блок-схеме алгоритма написать программу.
4. Отладить и выполнить программу.

Варианты заданий

1. Пусть имеется математическое выражение, в котором несколько уровней вложенных скобок, например 7-((X*((X+Y)/(J-3))+Y)/(4-2.5)). Необходимо удостовериться, что скобки расставлены правильно. Для этого надо убедиться в том, что:

1) число левых и правых скобок одинаково;

2) каждой правой (закрывающей) скобке предшествует левая (открывающая) скобка;

3) предусмотреть обработку трех типов скобок – [], {}, ().

2. Написать функцию сортировки элементов стека по возрастанию.

3. Написать функцию сортировки элементов стека по убыванию.

4. Написать программу, вычисляющую выражение, записанное в инфиксной форме. Следует воспользоваться двумя стеками – одним для операндов и другим для операторов. Вычислять инфиксное выражение, не преобразовывая его сначала в постфиксную форму.

5. Написать программу, считывающую входную строку в инфиксной форме и преобразующую ее в префиксную форму.

6. Написать программу, преобразующую строку в префиксной форме в строку в постфиксной форме.

7. Написать программу, преобразующую строку в постфиксной форме в строку в префиксной форме.

8. Написать программу, преобразующую строку в префиксной форме в строку в инфиксной форме.

9. Написать программу, преобразующую строку в постфиксной форме в строку в инфиксной форме.

10. Написать программу вычисления арифметического выражения с помощью стека.

Л а б о р а т о р н а я р а б о т а №4

Динамические структуры данных. Очередь, кольцо

Цель работы: изучить принципы построения динамических структур данных, организацию данных в виде очереди и кольца; разработать алгоритмы и написать программы для работы с очередью и кольцом.

Теоретические сведения

Очередь – список, который организован таким образом, что новые элементы добавляются в конец списка, а извлекаются из начала, т.е. организованный по принципу «первым пришел – первым вышел» (FIFO – First In, First Out). Для реализации очереди можно применять линейный связанный список с двумя внешними указателями на начало и конец, как показано на рис. 5.

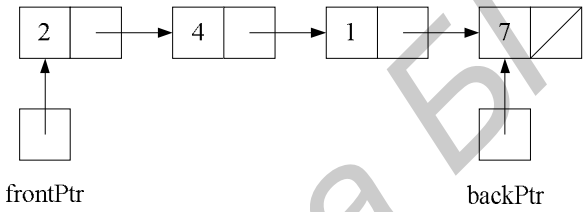

Рис. 5. Реализация очереди в виде линейного связанного списка с двумя внешними указателями

В кольцевом списке последний элемент содержит указатель на первый элемент списка (рис. 6).

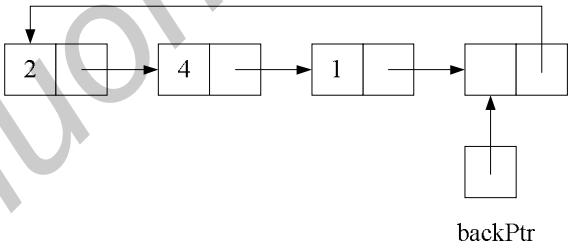

Рис. 6. Кольцевой связанный список с одним внешним указателем

Наряду с приведенной выше однонаправленной организацией списка может быть использована и двунаправленная организация. В этом случае движение по списку может быть выполнено в обоих направлениях.

Пример. Рассмотрим реализацию простейших операций при работе с однонаправленной очередью посредством двух внешних указателей (на «голову» и «хвост» очереди):

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
```

```
#include <string.h>
struct zap
{ char inf[50];                  // информационное поле
  zap *nx;                       // адресное поле
};
void add(zap **,zap **);
void del(zap **,zap **);
void del_any(zap **,zap **,char *);
void see(zap *);
void sort(zap **);
void main(void)
{ zap *h,*t;       // указатели на голову и хвост очереди
  char l,*st;
  st=(char *)malloc(10);
  h=t=NULL;
  while(1)
  { puts("вид операции: 1– создать очередь");
    puts("                2 – вывод содержимого очереди");
    puts("                3 – удаление элемента из очереди");
    puts("                0 – окончить");
    fflush(stdin);
    switch(getch())
    { case '1': add(&h,&t); break;       // добавление в хвост очереди
      case '2': see(h); break;           // просмотр с головы очереди
      case '3': if(h) del(&h,&t); break; // удаление с головы очереди
      case '0': return;
      default: printf("Выбран ошибочный режим, повторите \n");
    }
  }
}
// функция создания очереди
void add(zap **h,zap **t)
{ zap *n;
  puts("Создание очереди \n");
  do
  { if(!(n=(zap *) calloc(1,sizeof(zap))))
    { puts("Нет свободной памяти");
      return;
    }
    puts("Введите информацию в inf");
    scanf("%s",n->inf);
    if (!*h)              // очередь еще не создана
```

```
      *h=*t=n;          // устанавливаем оба указателя (голова и хвост)
                        // на единственный элемент очереди
    else                // очередь уже создана
    { (*t)->nx=n;       // добавляем очередной элемент в очередь
      *t=n;             // передвигаем указатель на хвост
    }
    puts("Продолжить (y/n): ");
    fflush(stdin);
  } while(getch()=='y');
}
// функция вывода содержимого очереди
void see(zap *h)
{ puts("Вывод содержимого очереди \n");
  if (!h)                       // указатель NULL
  { puts("Очередь пуста");
    return;
  }
  do
  { printf("%s\n",h->inf);    // вывод текущего элемента
    h=h->nx;                  // переход к следующему
  } while(h);
  return;
}
// функция удаления первого элемента очереди
void del(zap **h,zap **t)
{ zap *p;
  if(*t==*h)                 // в очереди только один элемент
  { free(*h);                // удаляем единственный элемент очереди
    *t=*h=NULL;              // очередь пуста
    return;
  }
  p=(*h)->nx;                // p указывает на элемент следующий за первым
  free(*h);                  // удаление первого элемента из очереди
  *h=p;                      // перемещение указателя h на голову очереди
}
```


Порядок выполнения работы

1. Изучить краткие теоретические сведения.
2. Составить блок-схему алгоритма.
3. По разработанной блок-схеме алгоритма написать программу.
4. Отладить и выполнить программу.

Варианты заданий

1. Имеются две упорядоченные очереди (одно- или двунаправленная), элементами которых являются целые числа. Объединить эти очереди в одну упорядоченную очередь.

2. Имеются два упорядоченных кольца (одно- или двунаправленное), в которых могут быть одинаковые элементы. Объединить оба кольца в одно упорядоченное кольцо, исключив повторяющиеся элементы.

3. Создать однонаправленную (двунаправленную) очередь, элементами которой является структура, содержащая информацию об учащемся: фамилия, номер группы, указатель на стек оценок, полученных за сессию. Для очереди предусмотреть возможность ее дополнения, сортировки (по фамилии, по номеру группы), удаления информации (по учащимся, получившим три и более неудовлетворительные оценки), поиска и редактирования информации в очереди.

4. Реализовать предыдущую задачу для списка, имеющего организацию однонаправленного (двунаправленного) кольца.

5. Инвертировать однонаправленный список (элемент содержит, например, фамилию), т. е. первый элемент становится последним, второй предпоследним и т. д. Дополнительных массивов и списков не создавать.

6. Дана очередь. Одним из ее элементов является указатель на кольцо. Реализовать следующие операции с указанными списками: в заданное кольцо подсоединить новый элемент, удалить элемент, вывести на экран содержимое кольца, удалить элемент из очереди, элементы заданного кольца распределить по другим кольцам.

7. Имеется однонаправленное кольцо. Элементом кольца является указатель на стек (учебная группа) и число учащихся в группе. Элементом стека – указатель на текстовую информацию (фамилия). В кольце найти группу с минимальным количеством учащихся и добавить ее стек к стеку следующей группы. Выбранный элемент кольца удалить.

8. Разработать программу, моделирующую общежитие и организованную по принципу списка (двунаправленной очереди, элемент которой – этаж общежития), содержащего следующую информацию: номер этажа, общее количество мест на этаже, количество свободных мест и указатель на однонаправленную очередь, элемент которой – номер комнаты и указатель на двунаправленное кольцо, содержащее информацию о проживающих в комнате людях. Организовать следующие режимы работы программы: поиск свободного места в общежитии, заселение, перемещение из комнаты в комнату, выселение.

9. Организовать список (очередь) автостоянок города. В очереди указываются общее количество мест, количество свободных мест и список номеров машин на этой автостоянке. Обеспечить функционирование автостоянок: при появлении новой машины она ставится на свободное место, удаление машины со стоянки. Въезд-выезд со стоянки единственный. Предусмотреть возможность создания новой автостоянки. При постановке автомобиля на стоянку запрашивать номер нужной автостоянки (на которую хотим поставить автомо-

биль). При отсутствии мест машина ставится в очередь ожидания свободного места на соответствующей стоянке.

10. Имеется очередь с информацией о больницах. Элементами очереди являются количество всех мест в больнице, количество свободных мест и указатель на функцию, вычисляющую (условно) расстояние от больного до больницы. Программа ведет учет свободных мест и распределяет больных в ближайшие больницы. Реализовать операции приема и выписки больного: если 0 – больной выписывается, 1 – поступает в ближайшую больницу, задается фамилия больного. По запросу выдается информация о состоянии мест в больнице.

Л а б о р а т о р н а я р а б о т а №5

Бинарные деревья

Цель работы: изучить принципы построения динамических структур данных, организацию данных в виде бинарного дерева; разработать алгоритмы и написать программы для работы с бинарным деревом.

Теоретические сведения

Бинарное дерево – это конечное множество элементов, которое содержит один элемент, называемый корнем дерева, а остальные элементы организованы как два непересекающихся подмножества, каждое из которых само является бинарным деревом. Эти подмножества называются левым и правым поддеревьями исходного дерева. Каждый элемент бинарного дерева называется узлом дерева. На рис. 7 показан общепринятый способ изображения бинарного дерева.

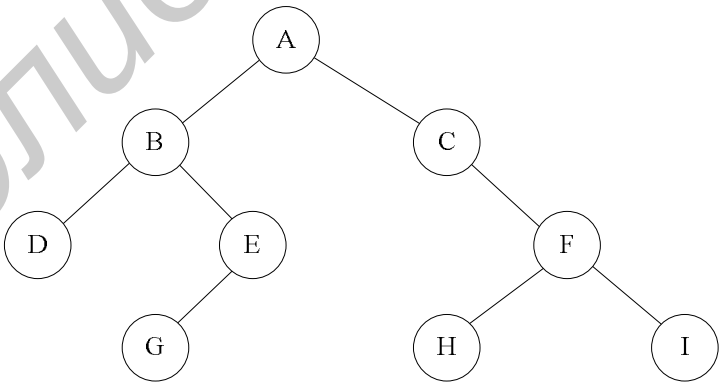

Рис. 7. Бинарное дерево

Здесь A – корень дерева, B (C) – корень левого (правого) поддерева. Иначе, узел A – отец, узлы B и C – его левый и правый сыновья. Узел, не имеющий сыновей, называется листом. Если каждый узел бинарного дерева, не являю-

щийся листом, имеет непустые правые и левые поддеревья, то дерево называется строго бинарным деревом. Строго бинарное дерево с n листами всегда содержит 2n – 1 узлов.

Бинарные деревья могут использоваться, например, для упорядоченного хранения очередной порции введенной информации.

Узел бинарного дерева можно описать в виде структуры, которая содержит информационную часть и указатели на левый и правый сыновние узлы:

```
struct node
{
     int key;                  //  ключ
      …                        //  данные
     struct node *left;        //  ссылка на левое поддерево (узел)
     struct node *right;       //  ссылка на правое поддерево (узел)
};
```

Для обхода дерева используют ключи – специальные переменные, облегчающие доступ к узлам дерева. Это некоторое значение, хранящееся в каждом узле дерева, причем все ключи в левом поддереве текущего узла меньше этого значения, а все ключи в правом поддереве – больше или равны ему.

Пример. При работе с бинарными деревьями возникают задачи создания дерева, добавления или удаления очередного узла, поиска узла с требуемой информацией, вывод информации, содержащейся в дереве.

```
#include <stdio.h>
#include <alloc.h>
#include <string.h>
#include <conio.h>
#include <dos.h>
#define N 20
#define M 20

struct der {char *inf;     // информационное поле*/
            int n;         // число встреч информационного поля в бинарном дереве*/
     struct der *l,*r;};   // указатель на левое и правое поддерево*/

void see_1(der *);
void see_2(der *);
der *sozd(der *);
void add(der *);
der *del(der *);

void main(void)
{ der *dr;
  dr=NULL;       // адрес корня бинарного дерева
  clrscr();
  while(1)
```

```
  { puts("вид операции: 1– создать дерево");
    puts("              2 – рекурсивный вывод содержимого дерева");
    puts("              3 – нерекурсивный вывод содержимого дерева");
    puts("              4 – добавление элементов в дерево");
    puts("              5 – удаление любого элемента из дерева");
    puts("              6 – выход");
    fflush(stdin);
    switch(getch())
    { case '1': dr=sozd(dr); break;
      case '2': see_1(dr); getch(); break;
      case '3': see_2(dr); getch(); break;
      case '4': add(dr); break;
      case '5': del(dr); break;
      case '6': return;
    }
    clrscr();
  }
}
// создание бинарного дерева
der *sozd(der *dr)
{ if (dr)
  { puts("Бинарное дерево уже создано");
    return (dr);
  }
  if (!(dr=(der *) calloc(1,sizeof(der))))
  { puts("Нет свободной памяти");
    getch();
    return NULL;
  }
  puts("Введите информацию в корень дерева");
  dr->inf=(char *) calloc(1,sizeof(char)*N);
  gets(dr->inf);
  dr->n=1;     // число повторов информации в дереве
  return dr;
}
// функция добавления узлов в бинарное дерево
void add(der *dr)
{ struct der *dr1,*dr2;
  char *st;              // строка для анализа информации
  int k;                 // результат сравнения двух строк
  int ind;
  if (!dr)
  { puts("Нет корня дерева \n");
```

```
   getch();
   return;
  }
  do
  { puts("Введите информацию в очередной узел дерева   (0 – выходl)");
   st=(char *) calloc(1,sizeof(char)*N);  //память под символьную информацию
   gets(st);
   if(!*st) return;                    // выход в функцию  main
   dr1=dr;
   ind=0;                              // 1 – признак выхода из цикла поиска
   do
   { if(!(k=strcmp(st,dr1->inf)))
     { dr1->n++;                       // увеличение числа встреч информации узла
       ind=1;                          // для выхода из цикла do ... while
     }
     else
     {if (k<0)                         // введ. строка  <  строки в анализируемом узле
       { if (dr1->l) dr1=dr1->l;  // считываем новый узел дерева
         else ind=1;                   // выход из цикла do ... while
       }
       else                            // введ. строка  >  строки в анализируемом узле
       { if (dr1->r) dr1=dr1->r;  // считываем новый узел дерева
         else ind=1;                  // выход из цикла do ... while
       }
     }
   } while(ind==0);
   if (k)       // не найден узел с аналогичной информацией
   { if (!(dr2=(struct der *) calloc(1,sizeof(struct der))))
     { puts("Нет свободной памяти");
       return;
     }
     if (k<0) dr1->l=dr2;   // ссылка в dr1 налево
     else dr1->r=dr2;       // ..... направо
     dr2->inf=(char *) calloc(1,sizeof(char)*N);
     strcpy(dr2->inf,st);   // заполнение нового узла  dr2
     dr2->n=1;
   }
   free(st);
  } while(1);       // любое условие, так как выход из цикло по return
}
// рекурсивный вывод содержимого бинарного дерева
void see_1(der *dr1)
{ if(dr1)
```

```
   // для вывода информации удалите комментарий с одной
   // из инструкций printf,  содержащихся ниже
  { //printf("узел содержит :   %s , число встреч %d\n",dr1->inf,dr1->n);
   if (dr1->l) see_1(dr1->l);     // вывод левой ветви дерева
   //printf("узел содержит :   %s , число встреч %d\n",dr1->inf,dr1->n);
   if (dr1->r) see_1(dr1->r);     // вывод правой ветви дерева
   //printf("узел содержит :   %s , число встреч %d\n",dr1->inf,dr1->n);
  }
}
// нерекурсивный вывод содержимого бинарного дерева,
// используя стек для занесения адресов узлов дерева
void see_2(der *dr1)
{ struct stek{ der *d;
            stek *s;} *st,*st1=NULL;
 int pr=1;
 for(int i=0;i<2;i++)   // в стек заносятся два элемента, содержащие указатель
 { st=(stek *)calloc(1,sizeof(stek));  // на корень дерева для прохода
  st->d=dr1;                           // по левому и правому поддеревьям
  st->s=st1;                           // указатель на стек вниз
  st1=st;
 }
  printf("узел содержит :   %s , число встреч %d\n",dr1->inf,dr1->n);
  while(st)
  { do
    { if(pr && dr1->l) dr1=dr1->l;    // переход на узел слева
      else if (dr1->r) dr1=dr1->r;    // переход на узел справа
      pr=1;                           // сброс принудительного движения вправо
      if(dr1->l && dr1->r)            // узел с двумя связями вниз
      { st1=st;
        st=(stek *)calloc(1,sizeof(stek));
        st->d=dr1;                    // указатель на найденый узел
        st->s=st1;                    // указатель на стек вниз
      }
      printf("узел содержит :   %s , число встреч %d\n",dr1->inf,dr1->n);
    } while(dr1->l || dr1->r);
     dr1=st->d;                 // возврат на узел ветвления
     st1=st->s;                 // в стеке адрес узла выше удаляемого
     free(st);                  // удаление из стека указателя на узел
                                // после прохода через него налево
     st=st1;
     if(dr1->r) pr=0;           // признак принудительного перехода
                                // на узел, расположенный справа от dr1, так как
                                // dr1->inf уже выведен при проходе слева
```

```
    }
}
// функция удаления узла дерева
der *del(der *dr)
{ struct der *dr1,*dr2,*dr3;
  char *st;              // строка для анализа информации
  int k;                 // результат сравнения двух строк
  int ind;
  if(!dr)
  { puts("Дерево не создано \n");
    return NULL;
  }
  puts("Введите информацию для поиска удаляемого узла");
  st=(char *) malloc(sizeof(char)*N);
  fflush(stdin);
  gets(st);                  //строка для поиска узла в дереве
  if(!*st) return NULL;        // выход в функцию  main
  dr2=dr1=dr;
  ind=0;                      // 1 – признак выхода из цикла поиска
  do                          // блок поиска удаляемого из дерева узла
  { if (!(k=strcmp(st,dr1->inf)))
       ind=1;                 // узел со строкой st найден
    if (k<0)                  // введ. строка  <  строки в анализируемом узле
    { if (dr1->l)
      { dr2=dr1;              // запоминаем текущий узел
        dr1=dr1->l;           // считываем новый левый узел дерева
      }
      else ind=1;             // выход из цикла do ... while
    }
    if (k>0)                  // введ. строка  >  строки в анализируемом узле
    { if (dr1->r)
      { dr2=dr1;              // запоминаем текущий узел
        dr1=dr1->r;           // считываем новый правый узел дерева
      }
      else ind=1;             // выход из цикла do ... while
    }
  } while(!ind);
  free(st);
  if (k)
  { puts("Требуемый узел не найден \n");
    getch();
    return dr;
  }
```

```
  else
  { k=strcmp(dr1->inf,dr2->inf);
    dr3=dr1;
    if (k<0)                  // удаляемая вершина < предыдущей
    { dr3=dr1->r;         // поиск ссылки NULL влево
      while(dr3->l) dr3=dr3->l;
      dr2->l=dr1->r;     // сдвиг ветви, начинающейся с адреса dr1->r, влево
      dr3->l=dr1->l;
    }
    else                      // удаляемая вершина > предыдущей
    { dr3=dr1->l;        // поиск ссылки NULL вправо
      while(dr3->r) dr3=dr3->r;
      dr2->r=dr1->l;    // сдвиг ветви, начинающейся с адреса dr1->l, вправо    }
      dr3->r=dr1->r;
    }
  }
}
```

Порядок выполнения работы

1. Изучить краткие теоретические сведения.
2. Составить блок-схему алгоритма.
3. По разработанной блок-схеме алгоритма написать программу.
4. Отладить и выполнить программу.

Варианты заданий

1. Написать рекурсивную функцию поиска узла бинарного дерева с максимальным значением (число с плавающей точкой). Ключ дерева – число с фиксированной точкой.

2. Дано бинарное дерево. Определить количество узлов на N-м уровне бинарного дерева (N ввести с клавиатуры).

3. Дано бинарное дерево, в узлах которого расположены целые числа величиной от 0 до 50 000. Числа, меньшие 3000, вывести на экран и удалить из структуры дерева.

4. Дано бинарное дерево, в узлах которого расположены строки до восьмидесяти символов. Создать стек и поместить в него все строки, содержащие хотя бы одну букву «А». Найденные строки удалить из бинарного дерева.

5. Дано бинарное дерево. В узлах бинарного дерева хранятся слова. Определить поддерево максимальной длины, в качестве ключа используется целое число. В найденном поддереве найти слово минимальной длины.

6. В узлах бинарного дерева содержатся имя файла и дата его создания. Удалить из дерева все записи с именами файлов, которые были созданы до даты, заданной с клавиатуры.

7. В узлах бинарного дерева имеется элемент, определяющий частоту обращения к узлу. Создать новое дерево таким образом, чтобы путь к узлам был кратчайшим (рекурсию не использовать). В качестве ключа использовать частоту обращений к узлу. Частоты в узлах дерева не совпадают. В дереве не более пятисот узлов.

8. С помощью бинарного дерева вычислить арифметическое выражение. В арифметическом выражении используются операции +, -, *, / и (,).

9. Создать бинарное дерево, элементами которого является структура, содержащая информацию об учащемся: фамилия, номер группы, указатель на стек оценок, полученных за сессию. Обеспечить возможность добавления информации, выборки ее по заданному критерию (по фамилии, по номеру группы), удаления информации (по учащимся, получившим 3 и более неудовлетворительные оценки), поиска и редактирования информации в дереве.

10. Преобразовать однонаправленную (двунаправленную) очередь (кольцо) в сбалансированное бинарное дерево.

Лабораторная работа №6

Файлы. Текстовые файлы

Цель работы: изучить структуру текстовых файлов, функции для работы с текстовыми файлами; разработать алгоритмы и написать программы для работы с текстовыми файлами.

Теоретические сведения

Файл – это именованный объект, хранящий данные (программа или любая другая информация) на каком-либо носителе. Файл, как и массив, – это совокупность данных. В отличие от массивов файлы располагаются не в оперативной памяти, а на жестких дисках или внешних носителях, файл не имеет фиксированной длины, т. е. может увеличиваться и уменьшаться. Перед работой с файлом его необходимо открыть, а после работы – закрыть.

Различают два вида файлов – текстовые и бинарные. Текстовые файлы представляют собой совокупность ASCII-символов. Эта последовательность символов разбивается на строки, каждая строка заканчивается двумя кодами: 13, 10 (0xD, 0xA). К текстовым файлам относятся, например, *.bat, *.c, *.asm.

Библиотека языка С содержит функции для работы как с текстовыми, так и с бинарными файлами.

Перед работой с файлом его необходимо открыть. Для этого используется функция fopen(), которая при успешном открытии возвращает указатель на структуру типа FILE, называемый указателем на файл. Эта структура связана с физическим файлом и содержит всю необходимую информацию для работы с ним (указатель на текущую позицию в файле, тип доступа и др.).

Функция открытия файла fopen() содержит два параметра, оба являются строковыми литералами.

FILE *fopen(char *filename, char *mode);

Первый параметр задает физическое местонахождение (путь) и имя открываемого файла, а второй – тип доступа к файлу.

Второй параметр может принимать следующие значения:

«r» – открыть файл для чтения;

«w» – открыть файл для записи. Если файл существует, то его содержимое теряется;

«a» – открыть файл для записи в конец файла. Если файл не существует, то он создается;

«r+» – открыть файл для чтения и записи. Файл должен существовать;

«w+» – открыть файл для чтения и записи. Если файл существует, то его содержимое теряется;

«a+» – открыть файл для чтения и записи в конец файла. Если файл не существует, то он создается.

К перечисленным комбинациям могут быть добавлены также «t» либо «b»:

«t» – открыть файл в текстовом режиме;

«b» – открыть файл в бинарном режиме.

Возможны следующие режимы доступа: «w+b», «wb+», «rw+», «w+t», «rt+» и др. Если режим не указан, то по умолчанию файл открывается в текстовом режиме.

После работы с файлом он должен быть закрыт функцией fclose(). Для этого необходимо в указанную функцию передать указатель на FILE, который был получен при открытии функцией fopen(). При завершении программы незакрытые файлы автоматически закрываются системой. Последовательность операторов для открытия и закрытия файлов следующая:

```
#include<stdio.h>
…
FILE *f;
If(!(f=fopen(“readme.txt”,”r+t”)))
{
      printf(“Невозможно открыть файл\n”); return;
}
…   // работа с файлом
fclose(f);
```

Рассмотрим функции, которые использует язык С для работы с текстовыми файлами. Для записи в файл и чтения из файла используются функции fprintf(), fscanf(), fgets(), fputs(). Отличие от функций printf(), scanf(), gets(), puts() состоит в том, что для работы с файлами добавлен параметр, который является указателем на структуру FILE.

После открытия каждый файл имеет так называемый указатель на текущую позицию в файле (УТПФ). Все операции над файлами (чтение и запись)

работают с данными, на которые указывает УТПФ. При каждом выполнении функции чтения или записи УТПФ смещается на количество записанных или прочитанных байт. Так реализуется последовательный доступ к данным.

Для организации чтения или записи данных в произвольном порядке необходимо реализовать установку указателя на некоторую заданную позицию в файле, для этого используется функция fseek().

int fseek(FILE *stream, long offset, int whence);

Параметр offset задает количество байт, на которое необходимо сместить указатель в направлении, указанном whence. Параметр whence может принимать следующие значения:

SEEK_SET или 0 – смещение выполняется от начала файла;
SEEK_CUR или 1 – смещение выполняется от текущей позиции указателя;
SEEK_END или 2 – смещение выполняется от конца файла.

Величина смещения может быть как положительной, так и отрицательной. Такой доступ к данным в файле называется произвольным.

Пример. Разработать программу ввода с клавиатуры чисел и добавления их в текстовый файл по возрастанию.

```
#include <stdio.h>
void main(void)
{  FILE *f;
   int a=5,b=7,c=10,d=13,e=14;
   int i,j;
   fpos_t l1,l2;                  // тип позиция в файле
   f=fopen("aaa","w+");
   fprintf(f,"%3d%3d%3d%3d%3d",a,b,c,d,e); // формат записи: 2 пробела 1 цифра
   while(1)
   { scanf("%d",&i);
     if (i==999) break;           // выход из программы
     rewind(f);                   // установка УТПФ в начало файла и сброс
     do                           // признака конца файла
     { fgetpos(f,&l1);            // УТПФ на начало поля считываемого числа
       fscanf(f,"%d",&j);
       if (feof(f) || j>i)  break;  // достигнут конец файла или считано число,
                                    // большее чем введенное с клавиатуры
     } while(1);
     rewind(f);
     if (j<i)                       // EOF и в файле нет числа > чем введенное
     {  fseek(f,0,2);               // выход на конец файла
        fprintf(f,"%3d",i);         // дозапись в конец файла
        continue;
     }
     fseek(f,-3,2);                 // УТПФ на последний элемент файла
     do
     { fgetpos(f,&l2);              // сдвиг всех чисел в массиве до
```

```
        fscanf(f,"%d",&j);       // l2 позиция УТПФ числа, которое > чем i
        rewind(f);
        l2+=3;
        fsetpos(f,&l2);
        fprintf(f,"%3d",j);
        fseek(f,l2-6,0);
      } while(l1<l2-3);
      fseek(f,l2-3,0);
      fprintf(f,"%3d",i);    // запись i на место числа, с которого
    }                        // произведен сдвиг всех чисел вниз
    fclose(f);
}
```

Порядок выполнения работы

1. Изучить краткие теоретические сведения.
2. Составить блок-схему алгоритма.
3. По разработанной блок-схеме алгоритма написать программу.
4. Отладить и выполнить программу.

Варианты заданий

1. Дан символьный файл f. Определить, сколько в файле f имеется слов, состоящих из одного, двух, трех и т. д. символов.

2. Даны текстовый файл f, символьная строка s. Получить все строки файла f, содержащие в качестве фрагмента строку s.

3. Дан символьный файл f. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Удалить из файла все однобуквенные слова и лишние пробелы. Результат записать в файл g.

4. Дан текстовый файл f. Зафиксируем натуральное k и перестановку чисел 1, …, k (ее можно задать с помощью последовательности натуральных чисел p_1, …, p_k , в которую входит каждое из чисел 1, …, k). При шифровке в исходном тексте в каждой из последовательных групп по k символов применяется фиксированная перестановка. Пусть k = 4 и перестановка – 3, 2, 4, 1. Тогда группа символов s_1, s_2, s_3, s_4 заменяется на s_3, s_2, s_4, s_1. Если в последней группе меньше четырех символов, то к ней добавляются пробелы. Используя изложенный способ:

а) зашифровать текст;

б) расшифровать текст.

5. Дан символьный файл f, содержащий произвольный текст. Слова в тексте разделены пробелами и знаками препинания. Получить n наиболее часто встречающихся слов и число их появлений.

6. Даны два символьных файла f1 и f2. Файл f1 содержит произвольный текст. Слова в тексте разделены пробелами и знаками препинания. Файл f2 содержит не более 40 слов, которые разделены запятыми. Эти слова образуют па-

ры: каждое первое слово считается заменяемым, каждое второе слово – заменяющим. Найти в файле f1 все заменяемые слова и заменить их на соответствующие заменяющие. Результат поместить в файл g.

7. Дан символьный файл f, компоненты которого являются целыми числами. Никакая из компонент файла f не равна нулю. Числа в файле идут в следующем порядке: десять положительных, десять отрицательных и т. д. Переписать компоненты файла f в файл g так, чтобы в файле g числа шли в следующем порядке: пять положительных, пять отрицательных и т. д.

8. Дан текстовый файл. Рассортировать в строках слова по величине их длины и записать в тот же файл.

9. Рассматриваются слова, содержащиеся в символьных файлах f1 и f2. Выяснить, сколько раз каждое из слов файла f2 встречается в файле f1.

10. В командной строке указано имя текстового файла. В файле записаны строки. Найти строку символов, не содержащую ни одного символа, входящего в другие строки.

Лабораторная работа №7

Бинарные файлы

Цель работы: изучить структуру бинарных файлов, функции для работы с бинарными файлами; разработать алгоритмы и написать программы для работы с бинарными файлами.

Теоретические сведения

Бинарные файлы – это файлы, которые не имеют структуры текстовых файлов. Каждая программа для своих бинарных файлов определяет собственную структуру.

Для чтения и записи в бинарный файл используются функции fwrite(), fread(), эти функции без каких-либо изменений копируют блок данных из оперативной памяти в файл и из файла в оперативную память соответственно. Такой способ записи – чтения требует меньших затрат времени, и его целесообразно использовать и для текстовых файлов, если работать с блоками информации. Функции имеют следующий формат:

unsigned fread(void *ptr, unsigned size, unsigned n, FILE *stream)

unsigned fwrite(void *ptr, unsigned size, unsigned n, FILE *stream)

здесь

size – размер блока информации в байтах;

n – количество блоков;

*stream – указатель на структуру FILE открытого файла.

Первым параметром передается указатель на буфер, в который будут помещены данные из файла функцией fread() или из которого данные будут прочитаны в файл функцией fwrite().

Следует отметить, что нет способа различать бинарные и текстовые файлы: любой бинарный файл можно открыть для работы как текстовый и наоборот. Однако это может привести к ошибкам, следовательно, необходимо работать с файлом в том режиме, в котором он был открыт.

Язык С обеспечивает возможность работы с файлами с помощью дескрипторов. Дескриптор – это целое число, которое система ставит в соответствие открытому файлу и в дальнейшем использует в операциях работы с файлами. Существует целый ряд функций работы с файлами через дескриптор. Например:

int fileno(FILE *stream); – функция возвращает дескриптор файла, связанный с указателем на файл stream;

long filelength(int handle); – функция возвращает размер файла в байтах. Для определения длины файла необходимо выполнить следующие действия:

```
void main(void)
{
      int ds;              // дескриптор файла
      FILE *fl;            // указатель на файл
      long len;            // длина файла
      ds=fileno(fl);       // дескриптор файла
      len= filelength(ds); // длина файла
}
```

Пример. Записать в бинарный файл числа и выполнить их сортировку методом «пузырька».

```
#include <stdio.h>
void main(void)
{ FILE *f;
  fpos_t n1,n2,n3;
  int i1,i2,i3,m[]={1,7,4,1,4,1};
  if (!(f=fopen("aa","w+b")))
  { puts("файл не может быть создан");
    return;
  }
   fwrite(m,sizeof(int),sizeof(m)/sizeof(int),f);
  n3=sizeof(int);
  fseek(f,-n3,2);
  n3=ftell(f);
  rewind(f);
  n1=0;
  while(1)
  { if(n1>=n3-sizeof(int)) break;
    n2=n3;
   while(n1<n2)
```

```
   { fsetpos(f,&n2);      // УТПФ на позицию второго считываемого числа
     fread(&i2,sizeof(int),1,f);   // считываем второе число
     n2-=sizeof(int);
     fsetpos(f,&n2);      // УТПФ на позицию первого считываемого числа
     fread(&i1,sizeof(int),1,f); // считываем первое число

    if(i1>i2)                    // сравнение чисел i1 и i2
    {  fsetpos(f,&n2);           // замена чисел i1 и i2 в файле
       fwrite(&i2,sizeof(int),1,f);
       fwrite(&i1,sizeof(int),1,f);
    }
   }
   n1+=sizeof(int);
 }
 rewind(f);
 fread(m,sizeof(m),1,f);
 for(i1=0;i1<sizeof(m)/sizeof(int);printf("%3d",m[i1++]));
 fclose(f);
}
```

Порядок выполнения работы

1. Изучить краткие теоретические сведения.
2. Составить блок-схему алгоритма.
3. По разработанной блок-схеме алгоритма написать программу.
4. Отладить и выполнить программу.

Варианты заданий

1. Дан бинарный файл f. В файле содержится информация о жителях района (ФИО, жилая площадь и количество человек, проживающих на данной площади). Написать программу, которая:

– сортирует файл f по величине жилой площади, приходящейся на одного человека;

– добавляет, удаляет и выводит на экран информацию из файла.

2. Дан бинарный файл f, содержащий сведения об экспортируемых товарах: наименование товара; страна, импортирующая товар; объем поставляемой партии в штуках. Найти страны, в которые экспортируется данный товар, и общий объем его экспорта.

3. Имеются два бинарных файла, упорядоченных по возрастанию. Переписать информацию в третий файл, упорядочив ее по убыванию. Дополнительных массивов и файлов не использовать, сортировку не выполнять.

4. Дан бинарный файл f. В файле хранятся фамилия студента и список взятых им книг. Написать функции добавления студентов в файл, удаления из

файла. Вывести список книг, взятых указанным студентом, вывести фамилии студентов, которые не возвратили книги в указанный срок.

5. Дан бинарный файл, содержащий различные даты. Каждая дата – это число, месяц, год. Найти:

а) год с наименьшим номером;

б) все весенние даты;

в) самую позднюю дату.

6. Даны два бинарных файла f1 и f2. Файл f1 – это инвентарный файл, содержащий сведения о том, сколько изделий и какие виды продукции хранятся на складе. Файл f2 – это вспомогательный файл, содержащий сведения о том, насколько уменьшилось или увеличилось количество изделий по некоторым видам продукции. Вспомогательный файл может содержать несколько сообщений по продукции одного вида или не содержать ни одного такого сообщения. Обновить инвентарный файл f1 на основе вспомогательного файла f2.

7. В командной строке задается имя бинарного файла. Запись файла содержит ФИО и место жительства студента. Информацию из файла записать в массив. Написать функции сортировки записей:

– в алфавитном порядке по фамилии;

– в алфавитном порядке по месту жительства.

В файле осуществить поиск информации об указанном студенте.

8. В командной строке задается имя бинарного файла. В записях файла хранятся название магазина, его номер, адрес, а также имя файла, в котором хранится информация об имеющихся в нем книгах: количество, стоимость, издательство. Программа реализует следующие операции: выдачу справок, продажи, поступления новых книг, а также поиск требуемых книг.

9. В командной строке задается имя бинарного файла, в котором хранится информация о видеофильмах (название, тип фильма, год выпуска). Программа реализует следующие операции:

– добавляет в файл новую информацию;

– исключает информацию о заданных фильмах;

– выводит список фильмов соответствующего типа.

10. Из командной строки получить имя бинарного файла и границы диапазона чисел, которые нужно удалить из файла. Дополнительных массивов и файлов не использовать.

Литература

1. Керниган, Б. Язык программирования С / Б. Керниган, Д. Ритчи. – М. : Финансы и статистика, 1992.

2. Керниган, Б. Язык программирования С / Б. Керниган, Д. Ритчи. – Спб. : Невский диалект, 2004.

3. Юлин, В. А. Приглашение к С / В. А. Юлин, И. Р. Булатова. – Минск : Выш. шк, 1991.

4. Романовская, Л. М. Программирование в среде С (С++) для ПЭВМ / Л. М. Романовская, Т. В. Русс, С. Г. Свитковский. – М. : Финансы и статистика, 1992.

5. Шилд, Г. Программирование на Borland C++ / Г. Шилд. – Минск : ООО «Попурри», 1998.

6. Подбельский, В. В. Программирование на языке Си / В. В. Подбельский, С. С. Фомин. – М. : Финансы и статистика, 2000.

7. Вирт, Н. Алгоритмы и структуры данных / Н. Вирт. – СПб. : Невский Диалект, 2001.

8. Демидович, Е. М. Основы алгоритмизации и программирования. Язык Си / Е. М. Демидович. – Спб. : БХВ-Петербург, 2006.

9. Хусаинов, Б. С. Структуры и алгоритмы обработки данных. Примеры на языке Си / Б. С. Хусаинов. – М. : Финансы и статистика, 2004.

10. Кочан, С. Программирование на языке С / С. Кочан. – М. : Издат. дом «Вильямс», 2007.

Содержание

Св. план 2009, поз. 65

Учебное издание

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Лабораторный практикум
для студентов специальности
1-40 02 01 «Вычислительные машины, системы и сети»
всех форм обучения

В 2-х частях

Часть 2

Составители:
Луцик Юрий Александрович
Ковальчук Анна Михайловна
Лукьянова Ирина Викторовна
Бушкевич Алексей Владимирович

Редактор Е. Н. Батурчик
Корректор Л. А. Шичко
Компьютерная верстка Е. Г. Бабичева

Подписано в печать 23.12.2009. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Таймс». Отпечатано на ризографе. Усл. печ. л. 2,33.
Уч.-изд. л. 2,0. Тираж 150 экз. Заказ 28.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0494371 от 16.03.2009. ЛП №02330/0494175 от 03.04.2009.
220013, Минск, П. Бровки, 6