Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет непрерывного и дистанционного обучения

Кафедра экономической информатики

Контрольная работа №2
По курсу «Основы алгоритмизации и программирования» часть 2

Выполнил — Студент
Ф.И.О. Юруш Е.В.
№ зач. кн. (нет)

Проверил — Т. М. Унучек

Минск-2017

Содержание

Введение

Выполнение данной контрольной работы заключается в доработке консольного приложения разработанного при выполнении контрольной работы №1 таким образом, чтобы оно предусматривало использование возможности выбора пользователем способа хранения данных (текстовый формат или двоичный), очередей при реализации запросов ввода-вывода, оценки времени, необходимого на выполнение операций сортировки для каждого используемого способа. Предназначение данного приложения - автоматизация учёта продаж программного обеспечения. Цель данной контрольной работы - развитие практических навыков программирования на языке С и работы в среде программирования Visual C++, а также формирование компьютерной грамотности.

1. Теоретические вопросы

Поиск в строке. Алгоритм прямого поиска.

Начальные условия типичной задачи поиска подстроки в строке заключается в наличии массива S из N элементов и массива P из M элементов, при этом 0<M<=N. Суть задачи заключается в поиске вхождения массива P в массив N. Существует множество алгоритмов такого поиска. Выбор того или иного алгоритма зависит от следующих факторов: необходима ли оптимизация; количество запросов на поиск; требуется ли приблизительный поиск, одновременный поиск; размер алфавита и проч. Наиболее примитивным является алгоритм прямого поиска. Он заключается в последовательном сравнении символов подстроки со строкой и сдвигом подстроки вправо до тех пор пока конец подстроки не достиг конца строки, либо не будет найдено совпадение всех символов подстроки с символами строки поиска. Алгоритм прямого поиска является малозатратным и не нуждается в предварительной обработке данных и в дополнительном пространстве, но, большинство сравнений алгоритма прямого поиска являются лишними, поэтому он малоэффективен.

Бинарные деревья – основные понятия.

Бинарное дерево — иерархическая структура данных, в которой каждый элемент, называемый узлом, имеет не более двух ссылок на другие элементы структуры, называемые потомками. Первый элемент ещё называют родительским узлом или корнем, а потомки - левым и правым наследниками. Узел может быть пустым, либо состоять из данных и двух поддеревьев (каждое из которых может быть пустым или также иметь потомков). Если у узла оба поддерева пустые, то он называется листом. Каждый узел в дереве задаёт поддерево, корнем которого он же и является.

Бинарные деревья обычно используются для организации данных. В таком случае для каждого узла должно выполняться правило: в левом поддереве должны содержаться только ключи, имеющие значения, меньшие, чем значение данного узла, а в правом поддереве - только ключи, имеющие значения, большие, чем значение данного узла. Поиск элемента в такой структуре будет занимать меньше времени, чем в линейной структуре, он не требует перебора всех его элементов. Максимальное число шагов при поиске по бинарному дереву равно количеству уровней в такой структуре.

2. Практическая часть

Задание на разработку программы в соответствии с индивидуальным заданием: Модифицировать программную реализацию контрольной №1: предусмотреть возможность выбора пользователем способа хранения данных (текстовый формат или двоичный); предусмотреть использование очередей при реализации запросов ввода-вывода; предусмотреть оценку времени, необходимого на выполнение операций сортировки для каждого используемого способа.

Вариант индивидуального задания 1, предметная область «Учет продаж программного обеспечения.».

Листинг кода программы:

Создаём заголовочный файл header.h. В нём:

```
//объявляем константу в которой храним количество символов
//в поле название структуры Merchant
#define MAXFNAME 25
#define QMAX 3
```

```c
//объявляем структуру в котором будем хранить информацию о товаре
typedef struct tagMerchant
{
    int    merchNmb;            //номер в списке
    char   fname[MAXFNAME];     //название товара
    int    price;               //цена товара
    int    sold;                //количество проданных позиций

    //в этих переменных храним ссылку на следующую, предыдущую связанную структуру
    struct tagMerchant *pnext, *pprev;
} Merchant;
```

```c
//объявляем структуру в которой будем хранить очередь из трёх элементов
typedef struct tagQueue {
  Merchant qu[QMAX];
  int rear, frnt;
} queue;
```

```c
//объявляем прототипы функций используемых в программе
void print(Merchant *p);
void getUser(Merchant *p);
Merchant* addEnd(Merchant *p, Merchant *end, int stnumb);
void printList(Merchant *p);
void loadList(Merchant *p, char *file, Merchant **pend, Merchant **pbegin);
void saveList(Merchant *p, char *file);
int editStd(Merchant *p, Merchant **pend, Merchant **pbegin);
int deleteStd(Merchant *p, Merchant **pend, Merchant **pbegin);
void find(Merchant *p);
Merchant* insInv(Merchant *p1, Merchant *p2);
void insSort(Merchant *p, Merchant **begin, Merchant **end);
void signSort(Merchant *p, Merchant **begin, Merchant **end);
void exchSort(Merchant *p, Merchant **begin, Merchant **end);
void saveIntoFile(Merchant *p);
void saveFileMode(int n);
void loadListBinary(Merchant *p, char *file, Merchant **pend, Merchant **pbegin);
void saveListBinary(Merchant *p, char *file);
void printListQueue(Merchant *p, queue *q);
void flushQ(queue *q);
void insertQ(queue *q, Merchant *x);
void getUserQueue(queue *q, Merchant **begin, Merchant **end);
```

Создаём главный файл main.c. В нём:

```c
//подключаем необходимые заголовочные файлы стандартной библиотеки
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include <time.h>
//подключаем созданный заголовочный файл
#include "header.h"
//объявляем глобальную переменную типа clock_t
//из <time.h> для хранения значения времени
clock_t t;
```

```
//пишем главную функцию программы
void main()
{
    //объявляем переменные
    //переменная для выбранного пользователем номера команды
    int n=0, k=0;
    //перем. для временного хранения стр-ры, для передачи её из одной функции в другую
    Merchant std;
    //указатели на первый и последний элемент связанных структур
    Merchant *begin=NULL, *end=NULL;
    //объявляем указатель на структуру для организации очереди для ввода-вывода
    queue *q;
    //выделяем память для структуры queue по указателю q
    q = (queue*)malloc(sizeof(queue));
    //первый и последний элемент очереди выставляем нулевыми, что зн. что она пуста
    q->frnt = 0;
    q->rear = 0;
```

Организуем пользовательское меню.

```
    //пользовательское меню
    //1.добавить запись в список товаров 2.отредактировать запись в списке
    //3.сохр. список в файл 4.загруз. список из файла 5.вывести список на экран
    //6.выход 7.поиск строки по наименованию товара 8.сотртировка вставками
    //9.сортировка выделением 10.сортировка обменом 11.добавление нескольких записей
L:  printf("\n1.add    3.save   5.print  7.find     9.sortSign   11.add Queue\n2.edit\
     4.load   6.exit   8.sortInsr 10.exchSort\n");
    printf("Input the command number (from 1 to 7): ");
    //сохраняем введённое пользователем значение в переменной n
    scanf("%d", &n);

    //в зав-ти от введённого пользователем значения (1-7) запуск. соотв. функции,
    //при вводе '6' (6й команды) выходим без ошибки
    switch(n)
    {
    case 1: //принимаем значения от пользователя во временную структуру std
            getUser(&std);
            //связываем заполненную польз-ем стр-ру std с имеющимися структурами
            end=addEnd(&std,end,0);
            //при первом заполнении стр-ры указатели на 1 и последн. структуру равны
            if (begin==NULL) begin=end;  break;
    case 2: editStd(begin, &end, &begin); break; //ред-ем выбранную пользователем строку
    case 3: saveIntoFile(begin); break;//сохраняем список в файл
    case 4: //загружаем список из файла
            //загружаем форму сохранённой информации: бинарная, текстовая
            //и в зависимости от типа запускаем нужную функцию
            k=loadFileMode();
            if (k==1) loadList(begin,"list.dat",&end, &begin);
            else loadListBinary(begin,"list.dat",&end, &begin); break;
    case 5: printListQueue (begin, q); break;//печатаем список на экран
    case 6: exit(0); break;//завершаем программу без ошибки
    case 7: find(begin); break; //находим необходимую строку
    case 8: insSort(begin, &begin, &end);   break; //сортируем двусвязный список вставками
    case 9: signSort(begin, &begin, &end);  break; //сортируем двусвязный выделением
    case 10: //сортируем двусвязный список обменом (пузырьковая с-ка)
            exchSort(begin, &begin, &end);  break;
    case 11: getUserQueue(q, &begin, &end); break; //ввод очередью
    }
    //при вводе пользователем значения не равного 1-11
    //просим пользователя ввести команду ещё раз
    goto L;
}
```

Опишем функции которые запускаются при вводе команды добавления записи в список ‘1’.

```
//функция для приёма значений от пользователя
//вход параметр: указатель на структуру
void getUser(Merchant *p)
 {
     Merchant tmp;
     //поочерёдно запрашиваем и принимаем информацию от пользователя
     printf("\nInput merchant :");
     fflush(stdin);
     fgets(tmp.fname,MAXFNAME,stdin);
     //удаляем символ завершения строки кот. добавился функцией fgets
     tmp.fname[strlen(tmp.fname)-1]='\0';
     printf("Input sold amount: ");
     scanf("%d", &tmp.sold);
     fflush(stdin);
     printf("Input price: ");
     scanf("%d", &tmp.price);

     //указатели на пред. след. структуру укажем позже
     tmp.pnext = tmp.pprev = NULL;
     //копируем структуру tmp в структуру по указателю p (вход. пар-р)
     *p = tmp;
 }
```

```
//функция (Ф) для связывания заполненной польз-ем структуры с имеющимся списком
//структур. Ф принимает: заполненную структуру, указатель на последнюю
//структуру в списке, номер в списке который необходимо заполнить для
//добавляемого товара (т.к. данная ф-я еще используется при загрузке
//списка из файла). Ф возвращает: указатель на структуру который записывается в
//переменную end - указатель на последнюю структуру
Merchant* addEnd(Merchant *p, Merchant *end, int stnumb)
{
    //выделяем память для структуры, помещаем указатель на неё в pAdd
    Merchant *pAdd=(Merchant*)malloc(sizeof(Merchant));
    //заполняем память по адресу pAdd переданной, через параметр указатель p, структурой
    *pAdd=*p;
    if (end==NULL) //если добавляем первый элемент
    {
        //в указатель end помещаем указатель на заполненную структуру
        end=pAdd;
        //если переданный параметр stnumb равен нулю, тогда номеру в списке присваиваем 1
        //иначе значение равное переданному параметру stnumb
        if (stnumb==0) pAdd->merchNmb=1;
        else pAdd->merchNmb=stnumb;
    }
    else //если добавляем не первый элемент
    {
        end->pnext = pAdd;//связываем последний элемент с добавляемым
        pAdd->pprev=end;  //связываем добавляемый элемент с последним
        //если переданный параметр stnumb равен нулю, тогда номер в списке присваиваем
        //номер последней строки в списке плюс один
        if (stnumb==0) pAdd->merchNmb=end->merchNmb+1;
        else  pAdd->merchNmb=stnumb;
        end= pAdd;
    }
    //возвращаем указатель на последний элемент
    return end;
}
```

Опишем функции, которые запускаются при вводе команды редактирования записи в списке ‘2’.

```
//функция для редактирования выбранных пользователем строк
//вход. пар-ры: указатель на первую структуру,
//указатель на указатель последней в списке структуры,
//указатель на указатель первой в списке структуры,
int editStd(Merchant *p, Merchant **pend, Merchant **pbegin)
{    //объявляем переменные
     int n, k;
     Merchant *s, std;
     //выводим список на экран и предлагаем выбрать номер строки
     printList (p);
     printf("Input number of Merchant: ");
     scanf("%d", &n);
     //находим структуру номер в списке кот. равен выбранному
     while(p!=NULL) {
     if (p->merchNmb==n){     s=p; break;}
     p=p->pnext; }
     //предлагаем выбрать номер команды "что сделать со строкой"
L:   printf("\nWhat do you want to do with it? \n1.edit \n2.delete\
 \n3.exit to the main menu \nInput the command number : ");
     scanf("%d", &k);
     switch(k)
     {
     case 1: //при редактировании строки запрашиваем перевввести данные
             getUser(&std);//и сохраняем их в структуру std
             //в выбранную пользователем строку вносим изменения
             strncpy(s->fname,std.fname,sizeof(std.fname));
             s->price=std.price; s->sold=std.sold;
             goto M; //выход из цикла
     case 2: //удаляем строку функцией deleteStd, передаём в неё
             //указатель на удаляемую стр-ру, на первый и последний эл-ты списка
             deleteStd(s, pend, pbegin); goto M; //и выходим из цикла
     case 3: goto M; //выход из цикла в главное меню
     }
     goto L;  //переспрашиваем номер команды если введённой значение не 1,2,3
M:   return 0;//выходим без кода об ошибке
}
```

Опишем функцию по указателю deleteStd.

```
158  //удаление строки из списка, параметры идентичные функции editStd
159 int deleteStd(Merchant *p, Merchant **pend, Merchant **pbegin)
160  {
161      Merchant *pr=NULL, *pn=NULL;//указатели для пред. след. эл. списка
162      if (p->pnext==NULL) //если удаляемая строка последняя
163      {
164          pr=p->pprev; //сохраняем в pr указатель на предыд. эл. списка
165          if(pr!=NULL) //если предыд. эл. есть
166          {
167              pr->pnext=NULL; //обнуляем указатель в пред. эл-те
168              *pend=pr;       //предыдущий элемент объявляем последним
169
170          } else //если предыд. элемента нет (т.е. элемент в списке один)
171          {
172              *pend=NULL; //последний и первый элементы списка обнуляем
173              *pbegin=NULL;
174          }
175      }
176      else if (p->pprev==NULL) //если удаляемая строка первая
177      {
178          pn=p->pnext; //сохраняем в pn указатель на след. эл. списка
179          *pbegin=pn;  //указатель на первый элемент меняем на pn
180          pn->pprev=NULL; //в след эл-те обнуляем указатель на пред. эл-т
181      }
182      else //если удаляемая стр. ни первая ни последняя
183      {
184          pn=p->pnext; //сохраняем в pn указатель на след. эл. списка
185          pr=p->pprev; //сохраняем в pr указатель на предыд. эл. списка
186     //связываем указ-ли предыдущ. и след. эл-та так как тек. эл-т удаляем
187          pn->pprev=p->pprev;
188          pr->pnext=p->pnext;
189      }
190      free(p); //освобождаем память по указателю тек. элемента
191      p=NULL;  //обнуляем указатель
192      return 0; //выходим без кода ошибки
193  }
```

Опишем функцию, которая запускается при вводе команды сохранения списка в файл ‘3’.

```
//функция сохранения в файл с возможностью выбора
//формы хранения данных принимает указатель на
//первую структуру для сохранения
void saveIntoFile(Merchant *begin)
{
    int k;
    //меню для выбора пользователем формы хранения данных
L:  printf("\nInpur data storage format \n1.text \n2.binary\
 \n3.exit to the main menu \nInput the command number : ");
    scanf("%d", &k);
    switch(k)
    {
    case 1: saveFileMode(1); //запоминаем формат сохранения данных для загрузки
            saveList(begin,"list.dat"); //сохраняем в текстовом формате
            goto M;
    case 2: saveFileMode(2);
            saveListBinary(begin,"list.dat");//сохраняем в бинарном виде
            goto M; //и выходим из цикла
    case 3: goto M; //выход из цикла в главное меню
    goto L;  //переспрашиваем номер команды если введённой значение не 1,2,3
    }
M:  return 0;
}
```

```
//функция для сохранения режима записи:
//в текстовом виде или бинарном
void saveFileMode(int n)
{
    FILE *pf=fopen("filemode.dat","w");
    //если файл успешно открыт/создан
    if (pf!=NULL)
    {
        fprintf(pf,"%d", n);
        fclose(pf);//закрываем файл
    }
}
```

```c
//функция записи списка в файл
//параметр: указатель на первый элемент списка и название файла
void saveList(Merchant *p, char *file)
{
    //объявляем указатель pf типа FILE, открываем/создаём файл
    //функцией fopen (режим символьн. записи), и помещаем указатель на файл в pf
    FILE *pf=fopen(file,"w");

    //если файл успешно открыт/создан
    if (pf!=NULL)
    {
        while(p!=NULL)//проходимся по всем элементам списка
        {
        //и записываем элементы структур в файл
        fprintf(pf,"%d %s %d %d\n", p->merchNmb, p->fname, p->price, p->sold);
        p=p->pnext;
        }
        fclose(pf);//закрываем файл
    }
}
```

```c
//функция для сохранения информации в файл в бинарном виде,
//принимает указатель на первую структуру к записи и название файла
void saveListBinary(Merchant *p, char *file)
{
    int k=0, len;
    //объявляем указатель pf типа FILE, открываем/создаём файл
    //функцией fopen (режим символьн. записи), и помещаем указатель на файл в pf
    FILE *pf=fopen(file,"wb");

    //если файл успешно открыт/создан
    if (pf!=NULL)
    {
        //в начале файла оставляем место для числа равного числу элементов в файле
        fwrite(&k, sizeof(int), 1, pf);
        while(p!=NULL)//проходимся по всем элементам списка
        {
        //и записываем элементы структур в файл
        fwrite (&p->merchNmb, sizeof(int), 1, pf);
        //название товара записываем как размер названия в байтах
        //и байты самих символов названия
        len = strlen(&p->fname);
        fwrite(&len, sizeof(unsigned), 1, pf);
        fwrite(&p->fname, 1, len, pf);

        fwrite (&p->price, sizeof(int), 1, pf);
        fwrite (&p->sold,  sizeof(int), 1, pf);
        p=p->pnext;
        k++;
        }
        rewind(pf);//помещаем указатель в начало файла
        fwrite(&k, sizeof(int), 1, pf);//записываем число записанных элементов
        fclose(pf);//закрываем файл
    }
}
```

Опишем функцию, которая запускается при вводе команды загрузки списка из файла ‘4’.

```
//функция для чтения формата(текст., бинарный вид) сохранённой информации в файл
int loadFileMode()
{
    //объявляем указатель pf типа FILE, открываем/создаём файл
    //функцией fopen (режим символьн. чтения), и помещаем указатель на файл в pf
    FILE *pf=fopen("filemode.dat","r");
    int k;
    if (pf!=NULL)
    {
        while(!feof(pf))
        {
            fscanf(pf,"%d", &k);
        }
        fclose(pf);//закрываем файл
    }
    return k;
}
```

```
//функция загрузки списка из файла, параметры:
//указатель на структуру типа Merchant, название файла, указатель на
//указ-ль на первый эл-т списка, указ-ль на указ-ль на последний эл-т списка
void loadList(Merchant *p, char *file, Merchant **pend, Merchant **pbegin)
{
    Merchant tmp, *ptmp;//
    //объявляем указатель pf типа FILE, открываем/создаём файл
    //функцией fopen (режим символьн. чтения), и помещаем указатель на файл в pf
    FILE *pf=fopen(file,"r");
    //если файл успешно открыт/создан
    if (pf!=NULL)
    {
        //очищаем текущий список структур
        while(p!=NULL)
        {
            ptmp=p->pnext;
            deleteStd(p, *pend, *pbegin);
            p=ptmp;
        }
        *pend=NULL;
        *pbegin=NULL;
        //пока не достигнут конец файла читаем данные во временную структуру tmp
        while(!feof(pf))
        {
            fscanf(pf,"%d %s %d %d\n", &tmp.merchNmb, &tmp.fname, &tmp.price, &tmp.sold);
            tmp.pnext = tmp.pprev = NULL;
            //запускаем функцию addEnd, передаём в неё указатель на заполненную временную структуру,
            //указатель на указатель последнего элемента списка, и номер записываемого элемента в списке
            //функция добавит полученную из файла структуру в список
            *pend=addEnd(&tmp,*pend,tmp.merchNmb);
            //если элемент первый то последний элемент равен первому элементу
            if (*pbegin==NULL) *pbegin=*pend;
        }
        fclose(pf);//закрываем файл
    }
}
```

```
//функция загрузки информации из файла, сохранённой в бинарном виде, параметры:
//указатель на структуру типа Merchant, название файла, указатель на
//указ-ль на первый эл-т списка, указ-ль на указ-ль на последний эл-т списка
void loadListBinary(Merchant *p, char *file, Merchant **pend, Merchant **pbegin)
{   //объявляем шаблон структуры-буфера для загрузки, инициализируем его нулями
    //и указатель для очищения списка
    Merchant tmp={0,"",0,0}, *ptmp; int k=0,i,len;
    //объявляем указатель pf типа FILE, открываем/создаём файл
    //функцией fopen (режим чтения из файла данных в бинарном виде),
    //и помещаем указатель на файл в pf
    FILE *pf=fopen(file,"rb");
    //если файл успешно открыт/создан
    if (pf!=NULL)
    {   //очищаем текущий список структур
        while(p!=NULL)  {
            ptmp=p->pnext;
            deleteStd(p, *pend, *pbegin);
            p=ptmp; }
        //очищаем указатели на начало и конец
        *pend=NULL; *pbegin=NULL;
        //читаем число равное количеству сохранённых структур из списка
        fread(&k, sizeof(int), 1, pf);
        //читаем данные из файла и заполняем структуры пока не прочитаем все структуры
        for (i = 0; i < k; i++)
         {
            fread (&tmp.merchNmb, sizeof(int), 1, pf);  fread(&len, sizeof(int), 1, pf);
            fread(&tmp.fname, 1, len, pf);  fread (&tmp.price, sizeof(int), 1, pf);
            fread (&tmp.sold,  sizeof(int), 1, pf); tmp.pnext = tmp.pprev = NULL;
//запускаем функцию addEnd, передаём в неё указатель на заполненную временную структуру,
//указатель на указатель последнего элемента списка, и номер запис-го элемента в списке
//функция добавит полученную из файла структуру в список
            *pend=addEnd(&tmp,*pend,tmp.merchNmb);
            //если элемент первый то последний элемент равен первому элементу
            if (*pbegin==NULL) *pbegin=*pend;
        }
        fclose(pf);//закрываем файл
    }
}
```

Опишем функции, которая запускается при вводе команды печати списка на экран с использованием очереди ‘5’.

```
//функция для печати с использованием очереди
void printListQueue(Merchant *p, queue *q)
{   //выводим шапку таблицы
    printf("\n%s %s %s %s \n","# ","Merchant                    ","Amount","Price");
    printf("%s \n","-----------------------------------------");

    //циклом проходимся по элементам списка и выводим их на экран
    //ниже описанной функцией print
    while(p!=NULL)
    {
        insertQ(q, p);
        p=p->pnext;
    }
    flushQ(q);
}
```

```
//функция для добавления структуры в очередь
//принимает указатель на область памяти, где расположена очередь
//и указатель на добавляемую структуру
void insertQ(queue *q, Merchant *x) {
  int h;
    //добавляем в очередь только в случае если в очереди ещё есть место
    if(q->rear < QMAX-1)
  {
    //в очереди храним значения элементов структур
    strncpy(q->qu[q->rear].fname,x->fname,sizeof(x->fname));
    q->qu[q->rear].merchNmb=x->merchNmb;
    q->qu[q->rear].price=x->price;
    q->qu[q->rear].sold=x->sold;
    //увеличиваем номер хвоста очереди
    q->rear++;
  }
  //если очередь заполнилась то освобождаем её первый элемент
  if(q->rear==QMAX-1)
  { //печатаем первый элемент в очереди
    print(&q->qu[q->frnt]);
    //все последующие элементы сдвигаем в начало
    for(h = q->frnt; h < q->rear; h++)
    {
        strncpy(q->qu[h].fname,q->qu[h+1].fname,sizeof(q->qu[h+1].fname));
        q->qu[h].merchNmb=q->qu[h+1].merchNmb;
        q->qu[h].price   =q->qu[h+1].price;
        q->qu[h].sold    =q->qu[h+1].sold;
    }
    //уменьшаем номер элемента обозначающего конец рчереди
    q->rear--;
  }
}
```

```
//функция для освобождения очереди (все элементы очереди выводятся
//на печать)
void flushQ(queue *q)
{
    int h;
    //пока есть элементы в очереди, выводим их на печать
    while (q->rear!=q->frnt)
    {   //печатаем первый элемент
        print(&q->qu[q->frnt]);
        //смещаем все элементы к началу
        for(h = q->frnt; h<= q->rear; h++)
        {
            strncpy(q->qu[h].fname,q->qu[h+1].fname,sizeof(q->qu[h+1].fname));
            q->qu[h].merchNmb=q->qu[h+1].merchNmb;
            q->qu[h].price   =q->qu[h+1].price;
            q->qu[h].sold    =q->qu[h+1].sold;
        }
        //уменьшаем номер элемента обозначающего конец рчереди
        q->rear--;
    }
}
```

Опишем функцию, которая запускается при вводе команды поиска товаров по его наименованию, команда «7».

```
//функция поиска необходимых строк на экран, принимает указатель на первый элемент связанного списка
void find(Merchant *p)
{   //вводим переменную флаг(0,1) и переменную для хранения ввода пользователя
    int  flag=0;
    char tempname[MAXFNAME];
    //запрашиваем имя товара для поиска
    printf("\nInput name of merchant which you want to find:");
    scanf("%s", tempname);
    //циклом проходимся по элементам списка и выводим на экран строку
    //с информацией по необходимому товару
    while(p!=NULL)
    {
        if (strcmp(p->fname,tempname)==0)
        {
            if (flag==0)
            {
                //выводим шапку таблицы
                printf("\n%s %s %s %s \n","# ","Merchant                  ","Amount","Price");
                printf("%s \n","-----------------------------------------");
                flag=1;
            }
            print(p);
        }
        p=p->pnext;
    }
    //не нашли товар, сообщаем об этом пользователю
    if (flag==0)
    {printf("%s %s\n","There are no merchant called - ",tempname);}
}
```

Опишем функции сортировки, предусматривающие оценку времени выполнения, которые запускается при вводе команд 8-10.

Команда 8 – функция сортировки включением:

```
//функция для сортировки включением
void insSort(Merchant *p, Merchant **begin, Merchant **end)
{

    Merchant *pnext1, *pnext2;
    t=clock();
    //сдвигаем указатель на следующий элемент т.к. рассматривать будем всегда
    //текущий элемент и элемент левее его
    pnext1=p->pnext;
    while (pnext1!=NULL)
    {
        //запоминаем элемент следующий от двух рассматриваемых
        pnext2=pnext1->pnext;
        //пока рассматриваемый элемент не встанет на своё место в выравненном
        //в правильном порядке массиве, выполняем перстановку соседних элементов
        while (1)
            {
                //в случае если элемент стал первым в списке или если элемент
                //левее его меньше, заканчиваем делать перестановки
                if (pnext1->pprev==NULL) break;
                if (pnext1->price>pnext1->pprev->price) break;
                //корректируем указатель на начало или конец если необходимо
                if (pnext1->pprev->pprev==NULL) *begin=pnext1;
                if (pnext1->pnext==NULL) *end=pnext1->pprev;
                //сама функция перестановки
                pnext1=insInv(pnext1->pprev,pnext1);
            }
        //переставляем указатель на элемент следующий от двух рассматриваемых
        pnext1=pnext2;
    }
    t = clock() - t;
printf("\nOperation time: %d CPU cycles (%f seconds).\n",(int)t,((double)t)/CLOCKS_PER_SEC);
}
```

Вышеприведённая функция использует функцию перестановки элементов в списке:

```
//функция для смены местами элементов динамического массива
//работает как для смены соседних элементов так и для смены
//не соседних элементов
Merchant* insInv(Merchant *p1, Merchant *p2)
{
    Merchant *pnextTemp, *pprevTemp, *pprev2Temp, *pnextTemp2, *pprevTemp2;

    //здесь совершается смена указателей на объекты
    //таким образом делаем перестановку в двусвязном списке

    //при этом перестановка соседних элементов отличается от
    //перстановки не соседних, поэтому было разработано два блока

    if (p2->pprev==p1)//если элементы соседние
    {
    pprev2Temp=p1->pprev;
    pprevTemp=p1->pprev;
    pnextTemp=p2->pnext;

    p1->pnext=p2->pnext;
    p1->pprev=p2;

    p2->pnext=p1;
    p2->pprev=pprevTemp;

    if (pnextTemp!=NULL) pnextTemp->pprev=p1;
    if (pprev2Temp!=NULL) pprev2Temp->pnext=p2;
    }
    else//если элементы не соседние
    {
```

```
        pnextTemp=p2->pnext;
        pprevTemp=p1->pprev;
        pnextTemp2=p1->pnext;
        pprevTemp2=p2->pprev;

        p1->pnext=pnextTemp;
        p1->pprev=pprevTemp2;

        p2->pnext=pnextTemp2;
        p2->pprev=pprevTemp;

        pprevTemp2->pnext=p1;
        pnextTemp2->pprev=p2;
        if (pprevTemp!=NULL) pprevTemp->pnext=p2;
        if (pnextTemp!=NULL) pnextTemp->pprev=p1;
    }
    return p2;
}
```

Команда 9 – функция сортировки выбором:

```
//функция для сортировки выбором
void signSort(Merchant *p, Merchant **begin, Merchant **end)
{
    int n=0,k=0,m=0,i;
    Merchant *pSign=p, *pCount=p, *pBegin=p;
    //считаем все элементы и находим наименьший
    while(pCount!=NULL){
        if (pCount->price<pSign->price) pSign=pCount;
        pCount=pCount->pnext;
        k++;}
    //если наименший не первый в очереди двигаем его на первое место
    //путём перестановки с первый элементом
    if (pSign!=p)
    {
        if (pSign->pnext==NULL) *end=p;
        pBegin=insInv(p,pSign);
        *begin=pBegin;//корректируем указатель на начало очереди
    }
    //данный счётчик отражает количество элементов которые стоят в необходимом порядке
    n++;
    //проходится по циклу будем (число элементов минус один) раз
    //с учётом уже выставленного в верном порядке одного элемента
    while(n<k-1)
    {
        pCount=pBegin; pSign=pBegin; m=0;
        //в данном цикле мы находим наименьшее значение ключа из оставшихся элементов
        while(pCount!=NULL)
        {
            if (m<n) pSign=pSign->pnext;
            if (pCount->price<pSign->price && m>=n) pSign=pCount;
            pCount=pCount->pnext;
            m++;
        }
        pCount=pBegin;
        //здесь мы находим первый элемент после выставленных в верном порядке эл-тов
        for(i=0;i<n;i++) pCount=pCount->pnext;
        //если первый элемент после выставленных в верном порядке не наименьший
        //меняем его местами с наименьшим
        if (pSign!=pCount)
        {
            if (pSign->pnext==NULL)
            {   //корректируем указатель на последний элемент если необходимо
                *end=pCount;
            }
            //сама функция перестановки элементов
            insInv(pCount,pSign);
        }
        n++;//увеличиваем число элементов выставленных в правильном порядке
    }
    t = clock() - t;
printf("\nOperation time: %d CPU cycles (%f seconds).\n",(int)t,((double)t)/CLOCKS_PER_SEC);
}
```

Команда 10 – функция сортировки обменом:

```
//функция для сортировки обменом
//принимает указатель на первый элемент очереди
//указатель на указатель на первый элемент очереди
//указатель на указатель на последний элемент очереди
void exchSort(Merchant *p, Merchant **begin, Merchant **end)
{

    int k=0;
    Merchant *pnext1=p, *pnext2=p, *pbegin=p;

    //считаем количество элементов в очереди и помещаем значение в k
    while(pnext1!=NULL)
    {
        k++;
        pnext1=pnext1->pnext;
    }
    //сравниваем по два элемента и переставляем их местами в случае выполнения условия
    //проходимся по всей очереди таким образом k-1 раз
    while (k-1>0)
    {
        //каждую итерацию цикла начинаем сравнивать со 2го элемента и сравниваем его с предыдущим
        pnext1=pbegin->pnext;
        while (pnext1!=NULL)
        {
            pnext2=pnext1->pnext;

            if (pnext1->price<pnext1->pprev->price)
            {
                //корректируем указатели на начало и конец очереди, в случае необходимости
                if (pnext1->pprev->pprev==NULL) {*begin=pnext1; pbegin=pnext1;}
                if (pnext1->pnext==NULL) *end=pnext1->pprev;
                //сама перестановка элементов
                pnext1=insInv(pnext1->pprev,pnext1);
            }
            //двигаемся на следующий элемент от двух рассматриваемых
            pnext1=pnext2;
        }
        //уменьшем количество необходимых итераций цикла прохода по очереди
        k--;
    }
    t = clock() - t;
printf("\nOperation time: %d CPU cycles (%f seconds).\n",(int)t,((double)t)/CLOCKS_PER_SEC);
}
```

Опишем функции, которая запускаются при вводе команды ввода нескольких товаров одной командой с использованием очереди, команда «11».

```
//функция для добавления нескольких элементов в список с использованием очереди
void getUserQueue(queue *q, Merchant **begin, Merchant **end)
{
    Merchant std;
    int k1;
    //меню для пользователя для 1. повтора операции ввода элемента в список
    //или 2. выхода из операции ввода нескольких элементов с высвобожденим очереди
    //(передачей элементов из очереди в список)
L1: printf("\nYou want to add one more merchant \n1.yes \n2.no \nInput the command number : ");
    scanf("%d", &k1);
    switch(k1)
    {
    //операция ввода элемента в список через очередь на ввод
    case 1:
        {
            int h=0;
            //запрашиваем информацию у пользователя для добавления элемента
            getUser(&std);
            //добавляем элемент в очередь если есть место
            if(q->rear < QMAX-1)
             {
                strncpy(q->qu[q->rear].fname,std.fname,sizeof(std.fname));
                q->qu[q->rear].price=std.price;
                q->qu[q->rear].sold=std.sold;
                //указатель на следующий и предыдущий элементы в списке обнуляем
                q->qu[q->rear].pnext=NULL;
                q->qu[q->rear].pprev=NULL;
                //хвост очереди увеличиваем на один
                q->rear++;
             }
             //в случае если очередь заполнилась, высвобождаем первый элемент из очереди
             //и передаём его в список
              if(q->rear==QMAX-1)
              { //передаём первый элемент очереди в список
                *end=addEnd(&q->qu[q->frnt],*end,0);
                if (*begin==NULL) *begin=*end;
                //смещаем элементы очереди вперёд
                for(h = q->frnt; h < q->rear; h++)
                {
                    strncpy(q->qu[h].fname,q->qu[h+1].fname,sizeof(q->qu[h+1].fname));
                    q->qu[h].price   =q->qu[h+1].price;
                    q->qu[h].sold    =q->qu[h+1].sold;
                }
                //хвост очереди уменьшаем на один
                q->rear--;
              }
            goto L1;
        }
```

```
        //операция высвобождения очереди и выхода из операции
    case 2: {
                int h=0;
                //пока есть элементы в очереди, передаём их в список
                while (q->rear!=q->frnt)
                {   //передача первого элемента из очереди
                    *end=addEnd(&q->qu[q->frnt],*end,0);
                    if (*begin==NULL) *begin=*end;
                    //смещаем все элементы к началу
                    for(h = q->frnt; h<= q->rear; h++)
                    {
                        strncpy(q->qu[h].fname,q->qu[h+1].fname,sizeof(q->qu[h+1].fname));
                        q->qu[h].price   =q->qu[h+1].price;
                        q->qu[h].sold    =q->qu[h+1].sold;
                    }
                    //хвост очереди уменьшаем на один
                    q->rear--;
                }
                goto M2; //и выходим из цикла
            }
    }
    goto L1;  //переспрашиваем номер команды если введённой значение не 1,2
M2: return;
}
```

Интерфейс работы программы:

Выбор пользователем способа хранения данных (текстовый формат или двоичный)

```
1.add    3.save   5.print  7.find     9.sortSign   11.add Queue
2.edit   4.load   6.exit   8.sortInsr 10.exchSort
Input the command number (from 1 to 7): 3

Inpur data storage format
1.text
2.binary
3.exit to the main menu
Input the command number : 2
```

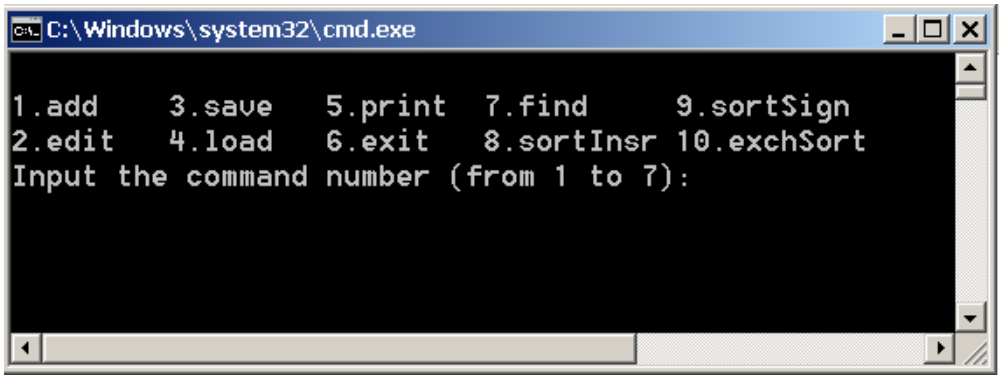
Запуск операции команды ввода нескольких товаров одной командой с использованием очереди:

```
1.add    3.save   5.print  7.find     9.sortSign   11.add Queue
2.edit   4.load   6.exit   8.sortInsr 10.exchSort
Input the command number (from 1 to 7): 11

You want to add one more merchant
1.yes
2.no
Input the command number : 1

Input merchant :Merchant1
Input sold amount: 10
Input price: 200

You want to add one more merchant
1.yes
2.no
Input the command number : 2
```

Оценка времени при выполнении операции сортировки:

```
1.add    3.save   5.print  7.find     9.sortSign   11.add Queue
2.edit   4.load   6.exit   8.sortInsr 10.exchSort
Input the command number (from 1 to 7): 10

Operation time: 0 CPU cycles (0.000000 seconds).
```

Главное меню:



Печать списка товаров:

```
#  Merchant                  Amount Price
----------------------------------------
1  PC Acer                   5       1000
2  Mouse Cannon              1       10
3  Keyboard 57x              2       30
4  PC 2Core                  1       900
```

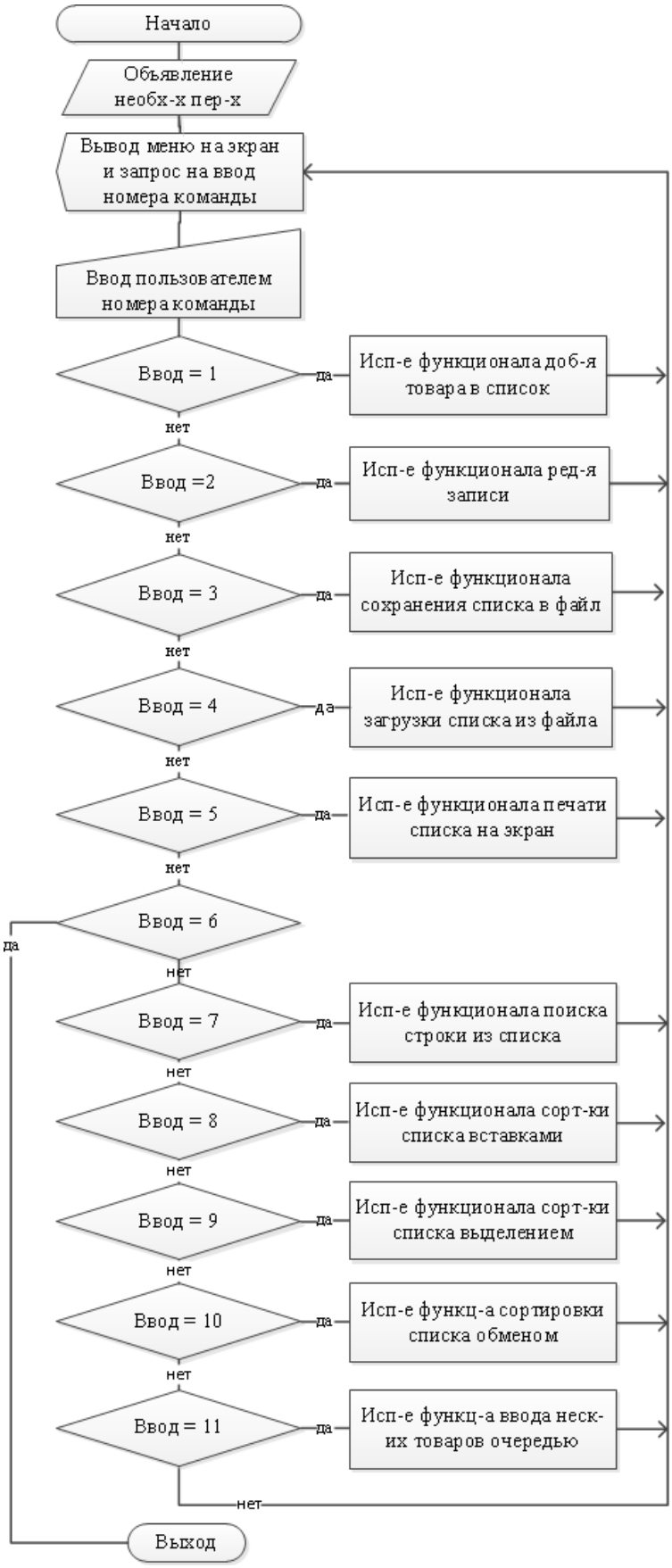
Редактирование записи о товаре:

```
1.add    3.save   5.print  7.find     9.sortSign
2.edit   4.load   6.exit   8.sortInsr 10.exchSort
Input the command number (from 1 to 7): 2

#  Merchant                  Amount Price
----------------------------------------
1  PC Acer                   5       1000
2  Mouse Cannon              1       10
3  Keyboard 57x              2       30
4  PC 2Core                  800     800
Input number of Merchant: 4

What do you want to do with it?
1.edit
2.delete
3.exit to the main menu
Input the command number : 1

Input merchant :PC 2Core
Input sold amount: 1
Input price: 900
```

Сортировка (по цене):

```
#  Merchant                  Amount Price
----------------------------------------
2  Mouse Cannon              1       10
3  Keyboard 57x              2       30
4  PC 2Core                  1       900
1  PC Acer                   5       1000
```

3. Блок-схема работы программы.



Заключение

В соответствии с индивидуальным заданием №2 доработана реализация консольной программы учёта продаж программного обеспечения, разработанной при выполнении контрольной работы №1. Отработаны навыки сохранения информации в файл в бинарном виде, использования очередей при реализации запросов ввода-вывода, оценки времени, необходимого на выполнение операций сортировки для каждого используемого способа. Предоставлены ответы на теоретические вопросы.

Литература

1. Подбельский В.В., Фомин С.С. Программирование на языке Си: Учебное пособие. 2-е доп. изд. - М.: Финансы и статистика – 600 с.: ил.
2. Березин Б.И., Березин С.Б. Начальный курс С и С++. –М.: Диалог - МИФИ, 1999 - 288 с.
3. Основы алгоритмизации и программирования. Язык Си: учеб. пособие / М. П. Батура, В. Л. Бусько, А. Г. Корбит, Т. М. Кривоносова. - Минск: БГУИР, 2007. - 240 с.: ил.