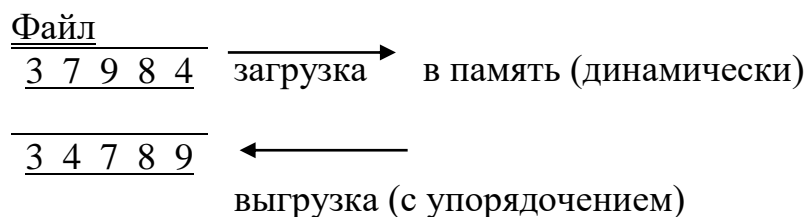
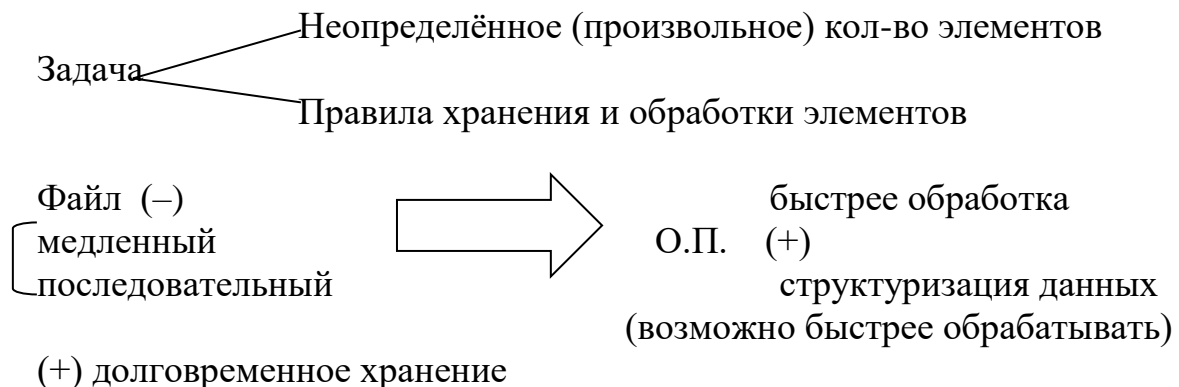


1. Постановка задачи:

- а) разработать способ хранения информации о студентах
- б) организовать в памяти динамическое размещение данных в виде направленного линейного списка,
- в) обеспечить эффективное выполнения операций по преобразованию списка,
- г) реализовать вывод данных о студентах, упорядоченных по некоторому признаку.



2. Анализ вариантов хранения и обработки информации.



3. Описание метода

- 1) Данные из файла (записанные в произвольном порядке, но по определённому формату представления информации по каждому студенту в отдельности) представлены при динамическом выделении для них памяти также по произвольным физическим адресам памяти, к примеру:
 - а) по физическим адресам памяти:

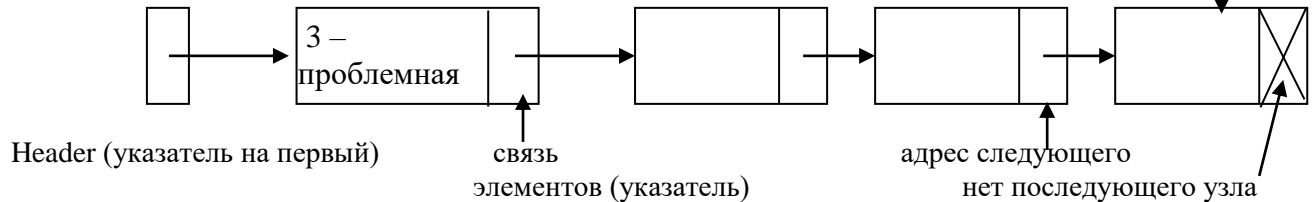
8		3		9		4		7	Память

б) по логической структуре пользователя

Начало

Логически:

проблемная часть



2) выполнение перестановки элементов списка для их следования друг за другом в выбранном пользователем порядке:

а) сортировка по возрастанию некоторого значения

б) сортировка по убыванию некоторого значения

в) следование информации в порядке, обратном исходному

Выполнение сортировки производится по одному из полей данных, описывающих студента.

3) вывод измененной информации (по порядку ее следования) в отдельный файл для хранения.

4. Описание алгоритма

Алгоритм решения задачи состоит из последовательности следующих вспомогательных алгоритмов

1) загрузка данных из файла

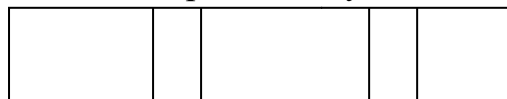
2) формирование списка элементов (данных о студентах) с изменением порядка следования

3) вывод в файл преобразованной информации

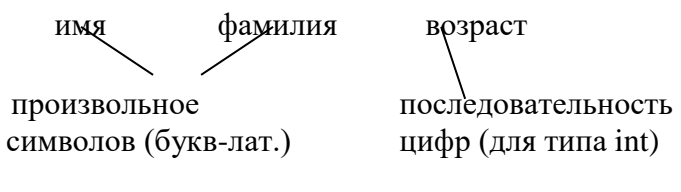
5. Структура данных

1) формат входного файла:

Построчно заносится информация об очередном студенте в следующей последовательности



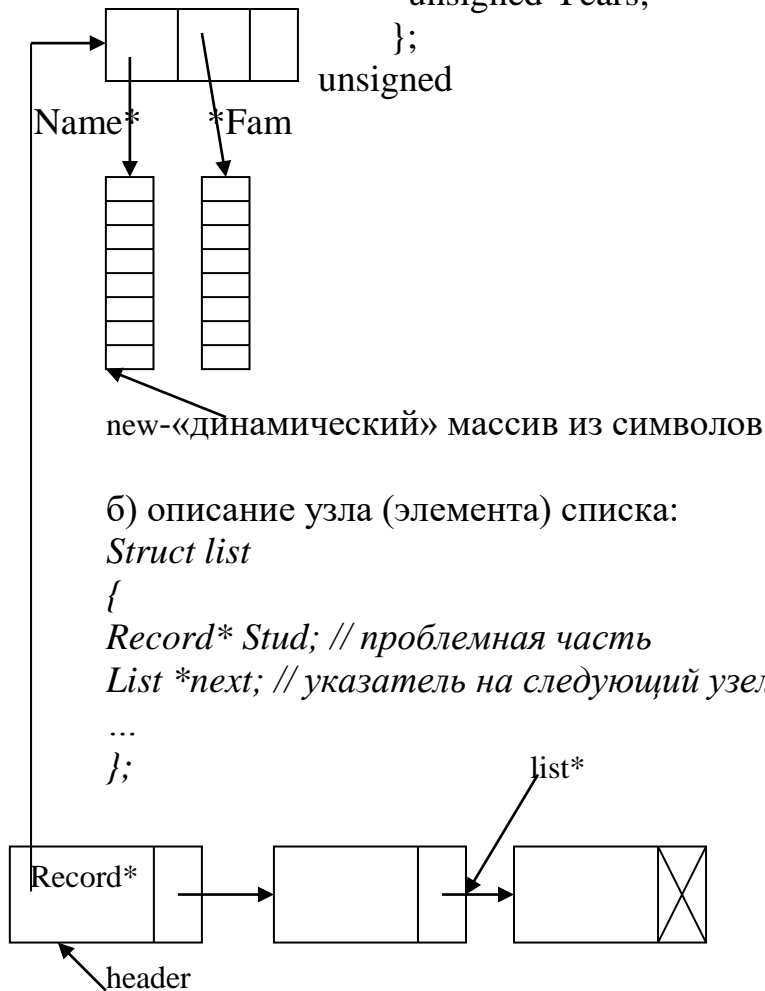
2) формат выходного файла



3) внутреннее представление информации о студенте с использованием динамически выделяемой памяти

а) представление проблемной части – описание студента:

```
struct Record
{
    char* Name;
    char* Fam;
    unsigned Years;
};
```



6. Операции с объектами (обслуживающие)

а) для проблемной части:

Конструктор: `Record::Record(fstream& f)` – осуществляет вычисление количества символов в имени и фамилии очередного студента, выделяет для них память и заполняет ее чтением из файла

Деструктор: `~Record () { delete Name; delete Fam; }` – освобождает выделенную память

б) для списка:

Конструктор: - сформировать проблемную часть

- сделать указатель на следующий Null

Деструктор: `~list()` – освободить память, занятую проблемной частью

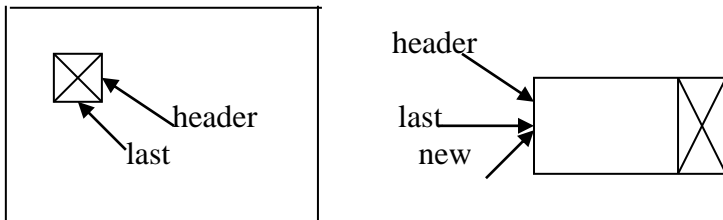
Операции (проблемные):

1) Создать пустой список:

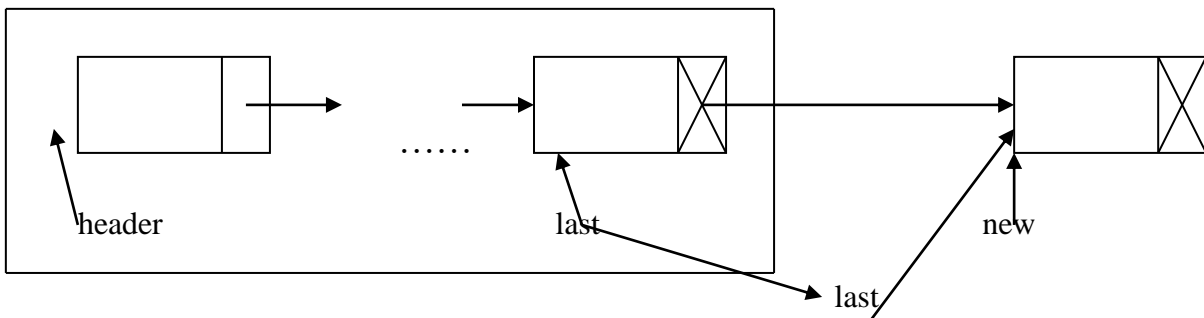
`Last=Header=NULL;`

2) Вставка нового узла в конец списка:

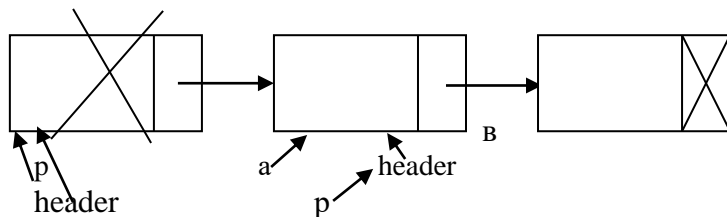
- а) new узел с записью в нём нового значения;
- б) указатель этого узла на следующий Null;
- в) к последнему (Last) поместить новый узел;
- г) сделать новый узел последним;
- д) last=header=первый (новый) адрес, сделать первый узел первым и последним.



Операция реализуется функцией-членом `list* list::Ins (.....)`; (см. п.9)



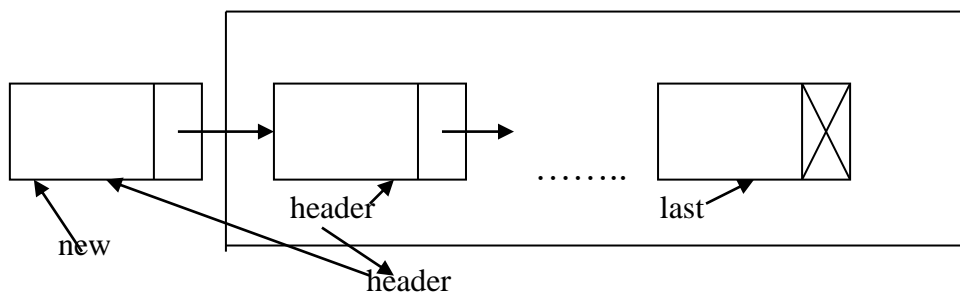
3) Удаление всего списка:



Результат: `header = NULL`

6
delete-вызов деструктора для проблемной части

4) Вставка нового узла в начало списка.



5) Вставка с сохранением упорядоченности значений проблемной части (по убыванию или по возрастанию). Возможны два подхода к решению этой задачи:

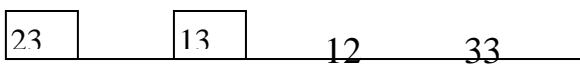
а) создать список и затем сортировать его;

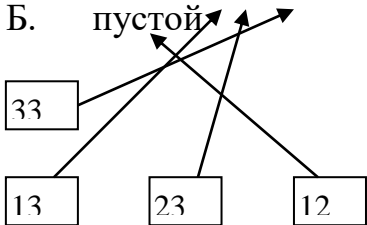
б) расставлять очередные элементы при их включении в список.

Для реализации после анализа быстродействия был выбран второй способ, подробное представление которого приводится в следующем п.7.

7. Оценка быстродействия:

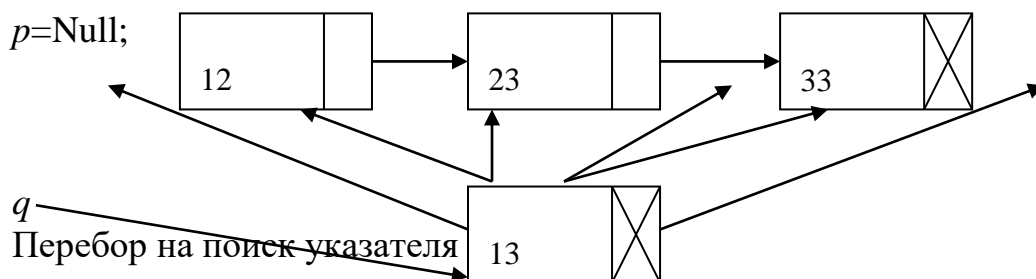
А. Задача: создать список и сортировать его: N-элементов- два действия (создать и сравнить)


 $(N-1)+(N-2)+\dots+1 \rightarrow \sim N^2 \text{ операций};$

Б. пустой  -0,1,2,...N-1 элементов

Особенности: 1) Сравнение при создании
2) Список упорядочен на каждом шаге

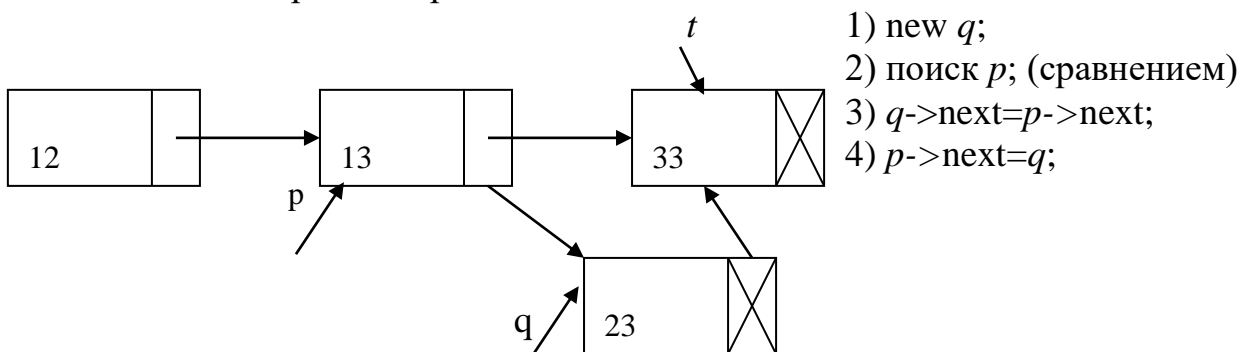
Определение места размещения и реализация вставки для варианта А:



Существует 3 варианта вставки (после элемента с указателем p):

- 1) вставка в начало ($p = \text{Null}$); (см. п.6.4)
- 2) вставка в конец ($p \rightarrow \text{next} = \text{Null}$); (см. п.6.2)
- 3) вставка в середину имеющегося списка ($p \neq \text{Null}$ и $p \rightarrow \text{next} \neq \text{Null}$).

Реализация алгоритма варианта 3:



Сортировка элементов списка при вставке выполняется по выбранному пользователем полю проблемной части из struct Record. В качестве критерия сортировки по полю Years выбирается сравнение числовых значений элементов.

При сортировке по полям Name и Fam критерием упорядочения элементов является лексико-графическое (по алфавиту) соответствие последовательности символов, занесенных в эти поля для разных элементов списка – для проверки таких слов используется функция strcmp из библиотеки <string.h>, реализующая сравнение слов.

8. Этап реализации задачи:

1.Выбор языка программирования

Для реализации созданного мной алгоритма, я выбрал согласно задания язык программирования C++. Технологические этапы работы с программой на C++ в системе программирования состоят из:

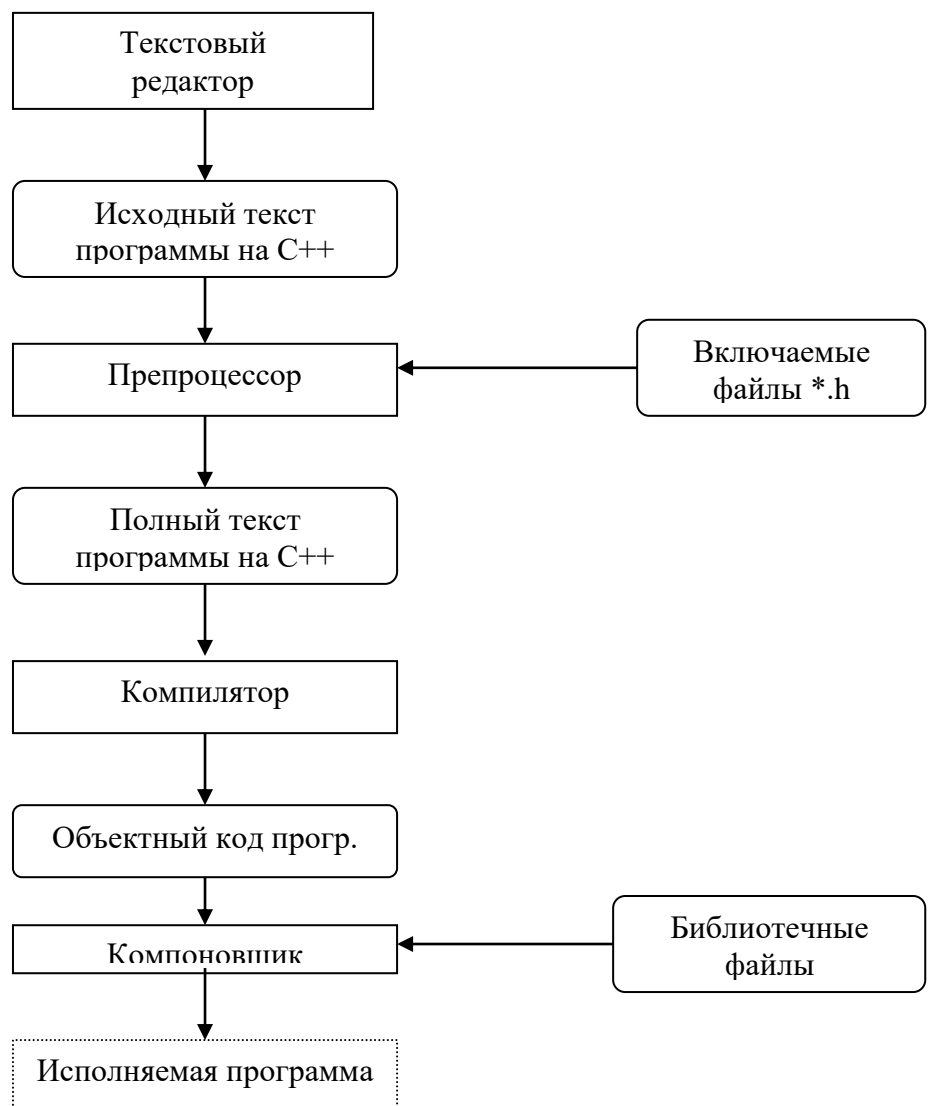


Рис. 8.1

На рис.8.1 прямоугольниками отображены программы – системные (сплошные линии) и пользовательская (пунктирные линии), а блоки с овальной формой обозначают файлы на входе и на выходе этих программ.

- 1) С помощью *текстового редактора* формируется текст программы и сохраняется в файле с расширением *crr*.
- 2) Осуществляется этап *препроцессорной обработки*, содержание которого определяется директивами препроцессора, расположенными перед заголовком программы (функции).
- 3) происходит *компиляция текста* программы на C++. В ходе компиляции могут быть обнаружены синтаксические ошибки, которые должен исправить сам программист.
- 4) Выполняется этап *компоновки с помощью* системной программы Компоновщик (Linker). Этот этап ещё называют *редактированием связей*. На данном этапе к программе подключаются библиотечные функции. В результате компоновки создаётся исполняемая программа в файле с расширением *exe*.

9. Представление алгоритмов решения задачи с использованием блок-схем.

А) Основной алгоритм решения задачи включает действия по:

- 1) выбору пользователем способа сортировки данных
- 2) вводу значений из файла с их включением в список и упорядочением
- 3) выводу результатов преобразований

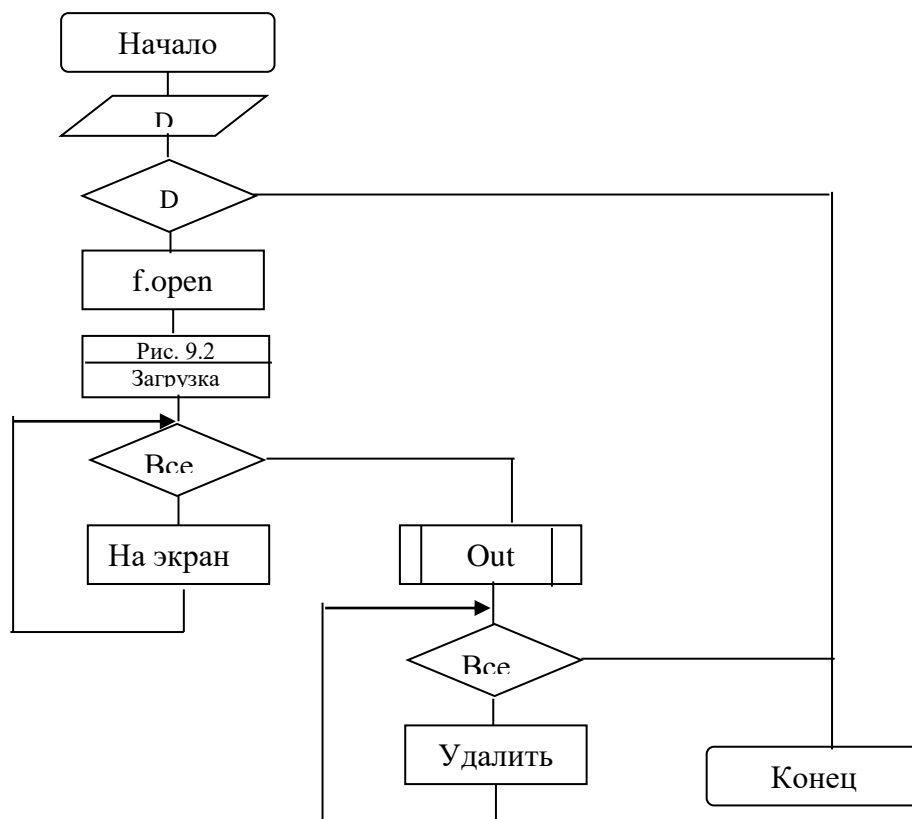


Рис. 9.1

Реализация алгоритма осуществлена в виде функции main.

Б) Выбранный алгоритм осуществляет последовательное чтение из файла записей об очередном студенте и вызов одной из функций преобразования списка для включения этого студента в список с сохранением выбранного порядка следования элементов. Эта последовательность представлена на рис. 9.2:

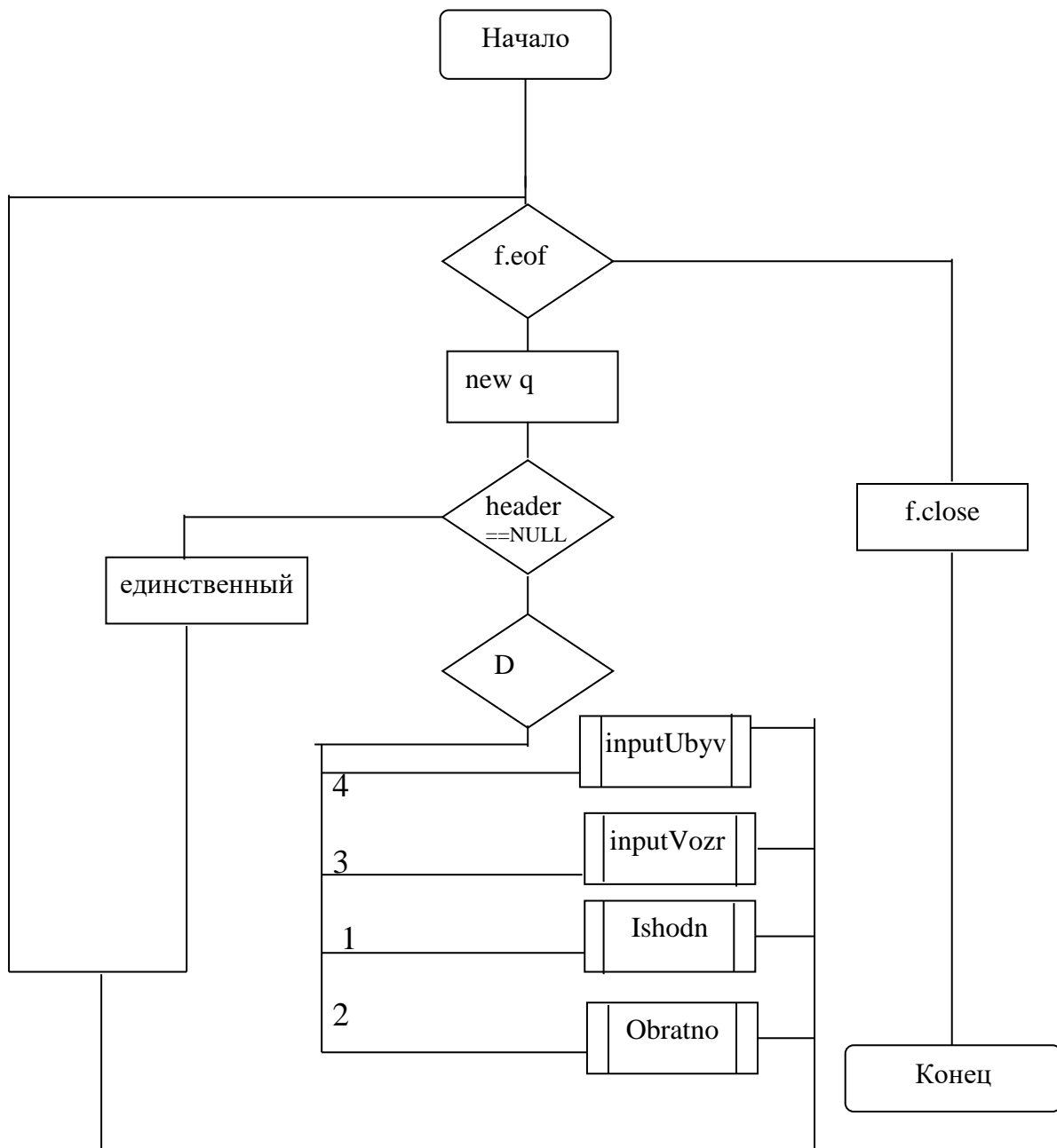


Рис.9.2

В) Общие алгоритмы по формированию списка, его сортировке при включении очередного элемента (по п.п.6 и 7) реализованы в виде функций-членов выбранных структур на C++. Пример для вставки с убыванием:

```

struct list
{
    .....
    list* inputUbyv (list*q) ;// вставляет в существующий список
    ...
};

```

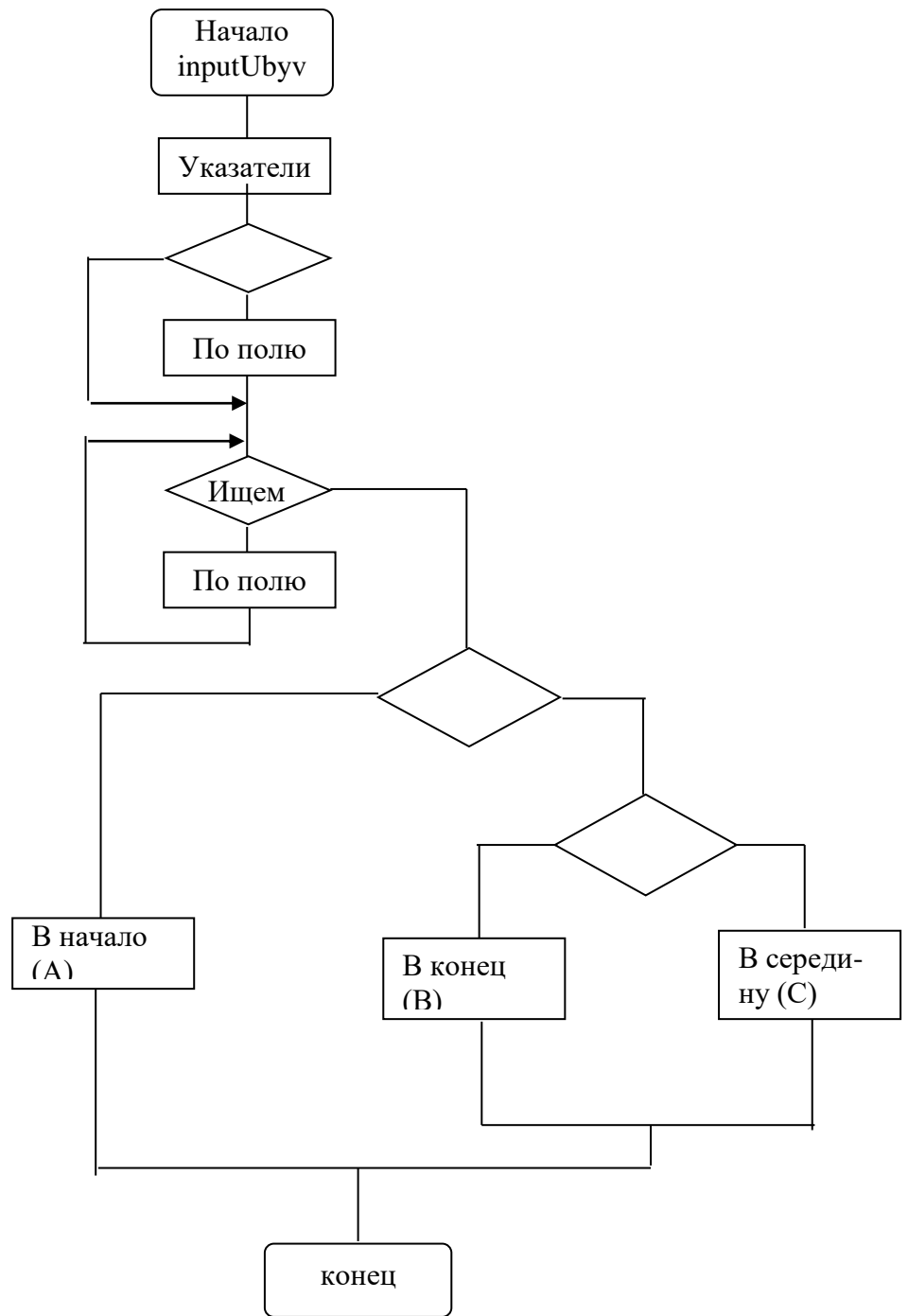



Рис. 9.3

неявно list *this

```

list* list::inputUbyv (list*q,int ZZ)
{ list *tmp, *t, *p;
  tmp=this ; // запомнить указатель 1-го элемента
  t=tmp; // начать с первого элемента перебор списка
  int bZnach;
  t=tmp; p=NULL;
  if(t!=NULL)
  { switch(ZZ)

```

обозначение указателя
на объект, для которого
вызвана функция,
указатель нельзя менять

```

    { case 1: bZnach=strcmp(t->Stud->Fam,q->Stud->Fam); break;
      case 2: bZnach=strcmp(t->Stud->Name,q->Stud->Name); break;
      case 3: if(t->Stud->Years>=q->Stud->Years) bZnach=1; else bZnach=-1; break;
    }
  }
while((bZnach>=0)&&(t!=NULL)) //2
{p=t;
 t=t->next;
 if(t!=NULL)
 {switch(ZZ)
 { case 1: bZnach=strcmp(t->Stud->Fam,q->Stud->Fam); break;
   case 2: bZnach=strcmp(t->Stud->Name,q->Stud->Name); break;
   case 3: if(t->Stud->Years>=q->Stud->Years) bZnach=1; else bZnach=-1; break;
 }
 }
 }
 if(p==NULL)
 {q->next=tmp; //A
  tmp=q;
 }
 else
 if(p->next==NULL)
 {p->next=q; //B
 }
 else
 {q->next=p->next; //C 3
  p->next=q; // 4
 }
 return tmp; // или старое значение указателя на 1-ый элемент
           // или в случае 1) новое значение (q) возвратит значение из tmp
}

```

Замечание: При реализации алгоритма при вставке в начало ($p = \text{Null}$), необходимо изменить указатель на 1-ый элемент, а функция вызвана именно для 1-го элемента, поэтому введен вспомогательный указатель tmp.

Вставка элементов в список при сортировке по возрастанию производится аналогично с помощью алгоритма, реализованного в функции-члене inputVozr, с изменением способа сравнения полей проблемной части. Формирование списка с сохранением исходного порядка следования реализуется функцией-членом Ishodn, а с записью в обратном порядке – Obratno. При этом представленный алгоритм упрощается, т.к. нет необходимости проводить поиск места вставки, которая осуществляется по значениям указателей Last и header соответственно.

Г) Для записи результатов преобразований используется вспомогательный алгоритм записи проблемной части элементов списка в файл, который реализуется функцией-членом Out:

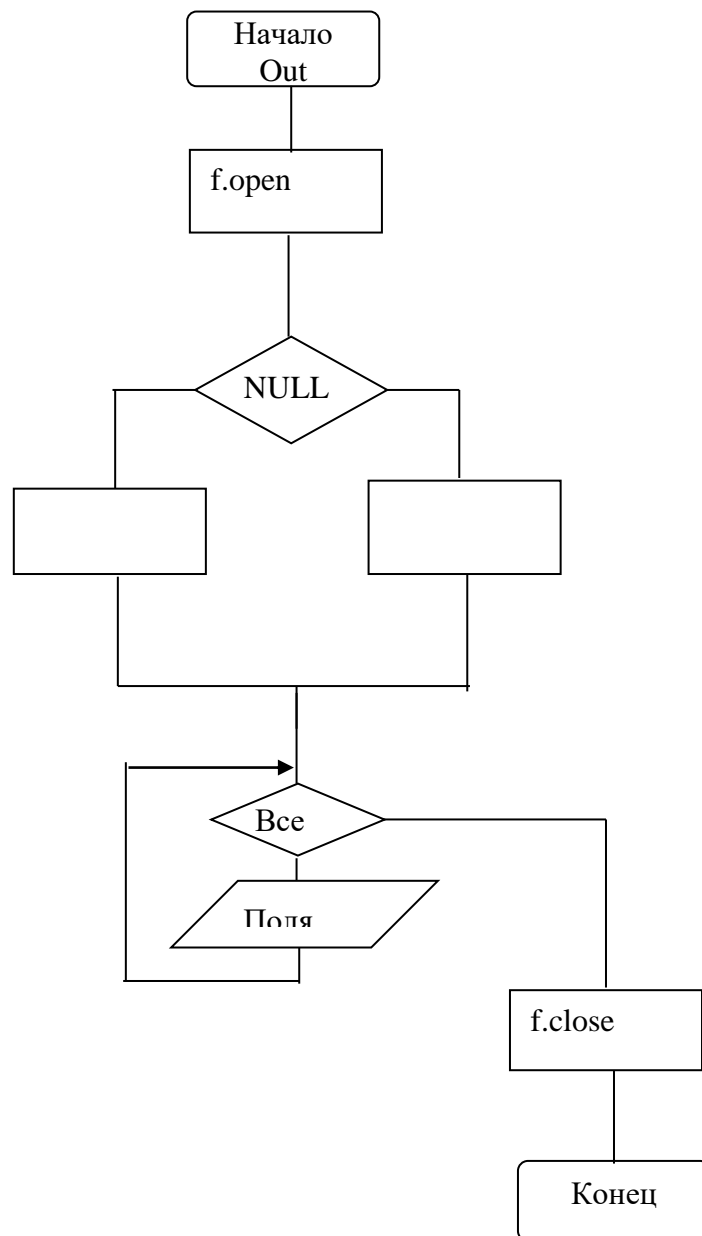


Рис.9.4

10. Программа на C++ :

```

#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

struct Record
{char* Name;
 char* Fam;
 unsigned Years;
 Record(fstream& f);
 ~Record() {delete Name; delete Fam;}
};

Record::Record(fstream& f)
{int i; char s;
```

```

long int fP;
i=0; fP=f.tellg();
do {f>>s; if(s==' ') break;
    i++;}
while(isalpha(s)&&(s!=' '));
Name=new char[i+1];
f.seekg(fP,ios::beg);
i=0;
do {f>>s; if(s==' ') break;
    Name[i]=s; i++;}
while(isalpha(s)&&(s!=' '));
Name[i]='\0';

i=0; fP=f.tellg();
do {f>>s; if(s==' ') break;
    i++;}
while(isalpha(s)&&(s!=' '));
Fam=new char[i+1];
f.seekg(fP,ios::beg);
i=0;
do {f>>s; if(s==' ') break;
    Fam[i]=s; i++;}
while(isalpha(s)&&(s!=' '));
Fam[i]='\0';

f>>Years; f.ignore(100,'\n');
}

```

```

struct list
{ //float x;
  Record* Stud;
  list* next;

  list(fstream& f)
  {next=NULL;
   Stud=new Record(f);
  }
  ~list()
  {cout<<"Delete znach="<< /*x<<*/ endl;
   delete Stud;
  }

  void Ishodn(list* q);
  list* Obratno(list* q);

  list* inputVozr(list* q,int ZZ);
  list* inputUbyv(list* q,int ZZ);

  void Out(char* name);
};

list* Last; // poslednii

void list::Ishodn(list* q)
{Last->next=q; // B)
 Last=Last->next; // r)
}

list* list::Obratno(list* q)
{q->next=this; // B)
 return q; // r)
}

```

```

}

list* list::inputVozr(list* q,int ZZ) // q - new
{list* tmp,*t,*p; // tmp - header , t - ocherednoy , p - pered t
int bZnach;
tmp=this;
t=tmp; p=NULL;
if(t!=NULL)
{switch(ZZ)
{case 1: bZnach=strcmp(t->Stud->Fam,q->Stud->Fam); break;
case 2: bZnach=strcmp(t->Stud->Name,q->Stud->Name); break;
case 3: if(t->Stud->Years<=q->Stud->Years) bZnach=-1; else bZnach=1;
break;
}
}
while((bZnach<=0)&&(t!=NULL)) //2
{p=t;
t=t->next;
if(t!=NULL)
{switch(ZZ)
{case 1: bZnach=strcmp(t->Stud->Fam,q->Stud->Fam); break;
case 2: bZnach=strcmp(t->Stud->Name,q->Stud->Name); break;
case 3: if(t->Stud->Years<=q->Stud->Years) bZnach=-1; else bZnach=1;
break;
}
}
}
if(p==NULL)
{q->next=tmp; //A
tmp=q;
}
else
if(p->next==NULL)
{p->next=q; //B
}
else
{q->next=p->next; //C 3
p->next=q; // 4
}
return tmp;
}

list* list::inputUbyv(list* q,int ZZ)
{list* tmp,*t,*p;
int bZnach;
tmp=this;
t=tmp; p=NULL;
if(t!=NULL)
{switch(ZZ)
{case 1: bZnach=strcmp(t->Stud->Fam,q->Stud->Fam); break;
case 2: bZnach=strcmp(t->Stud->Name,q->Stud->Name); break;
case 3: if(t->Stud->Years>=q->Stud->Years) bZnach=1; else bZnach=-1;
break;
}
}
while((bZnach>=0)&&(t!=NULL)) //2
{p=t;
t=t->next;
if(t!=NULL)
{switch(ZZ)
{case 1: bZnach=strcmp(t->Stud->Fam,q->Stud->Fam); break;
case 2: bZnach=strcmp(t->Stud->Name,q->Stud->Name); break;
case 3: if(t->Stud->Years>=q->Stud->Years) bZnach=1; else bZnach=-1;
break;
}
}
}

```

```

    }
}
}
if (p==NULL)
{q->next=tmp;    //A
 tmp=q;
}
else
if (p->next==NULL)
{p->next=q;      //B
}
else
{q->next=p->next; //C 3
 p->next=q;      // 4
}
return tmp;
}

void list::Out(char* name)
{list* t; int i=1;
 t=this;
 ofstream fout;
 fout.open(name);

 if(t==NULL) fout<<"pusto"<<endl;
 else fout<<" Fam " <<" Name " <<" Years"<<endl;
 while(t!=NULL)
 { //fout<<t->x;
  fout<<t->Stud->Fam<<" " <<t->Stud->Name<<" " <<t->Stud->Years<<endl;
  t=t->next;
  i++;
 }
 fout<<endl<<endl;
 fout.close();
}

void main()
{list *header=NULL,*q,*t; // q - new      t - ocherednoy
 int D,ZZ;
 float znach; fstream f;

 cout<<"Variant:"; cin>>D;
 if((D==3)|| (D==4)) {cout<<"Pole:"; cin>>ZZ;}

 if((0<D)&&(D<5))
 { f.open("input.txt",ios::in);
  if(!f) {cout<<"No input.txt"<<endl; exit(1);}
  // f>>znach;
  f.unsetf(ios::skipws);

  while(1)
  {if(f.eof()) break;

   q=new list(f); // 1)
   if(header==NULL)
   {header=q; Last=header;
    } // A
   else
   {switch(D)
    {case 1: header->Ishodn(q); break;
     case 2: header=header->Obratno(q); break;
     case 3: header=header->inputVozr(q,ZZ); break;
     case 4: header=header->inputUbyv(q,ZZ); break;
    }
   }
  }
}

```

```

    }
    //      f>>znach;
    }
    f.close();

//out floats

t=header; if(t==NULL) cout<<"pusto"<<endl;
while(t!=NULL)
{
    //cout<<t->x;
    cout<<t->Stud->Fam<<" "<<t->Stud->Name<<" "<<t->Stud->Years<<endl;
    t=t->next;
}
cout<<endl<<endl;

header->Out("out.txt");

//Delete memory floats

t=header;
while(t!=NULL)
{
    t=header->next;
    delete header;
    header=t;
}
}
cout<<"End of program"<<endl<<endl;
}

```

11. Тестирование

1) упорядочение по возрастанию и по фамилии

N	Данные во входном файле	Данные в памяти (q - жирное)	Данные в выходном файле
0	Kolya Fedin 23 Fedya Kolin 32 Anton Yarov 12 Ira Ari 45 Oly Mor 78		
1		Kolya Fedin 23	
2		Kolya <u>Fedin</u> 23 Fedya Kolin 32	
3		Kolya <u>Fedin</u> 23 Fedya <u>Kolin</u> 32 Anton Yarov 12	
4		Ira <u>Ari</u> 45 Kolya <u>Fedin</u> 23 Fedya <u>Kolin</u> 32 Anton <u>Yarov</u> 12	
5		Ira <u>Ari</u> 45 Kolya <u>Fedin</u> 23 Fedya <u>Kolin</u> 32 Oly Mor 78 Anton <u>Yarov</u> 12	Fam Name Years Ari Ira 45 Fedin Kolya 23 Kolin Fedya 32 Mor Oly 78

			Yarov Anton 12
--	--	--	----------------

2) убывание по имени

N	Данные во входном файле	Данные в памяти (q - жирное)	Данные в выходном файле
0	Kolya Fedin 23 Fedya Kolin 32 Anton Yarov 12 Ira Ari 45 Oly Mor 78		
1		Kolya Fedin 23	
2			
3			
4			
5			Fam Name Years Yarov Anton 12 Kolin Fedya 32 Ari Ira 45 Fedin Kolya 23 Mor Oly 78

3) в **обратном** порядке следования

N	Данные во входном файле	Данные в памяти (q - жирное)	Данные в выходном файле
0	Kolya Fedin 23 Fedya Kolin 32 Anton Yarov 12 Ira Ari 45 Oly Mor 78		
1		Kolya Fedin 23	
2		Kolya Fedin 23 Fedya Kolin 32	
3			
4			
5			Fam Name Years Mor Oly 78 Ari Ira 45 Yarov Anton 12 Kolin Fedya 32 Fedin Kolya 23

12. Вывод:

- а) реализована структура направленного линейного списка (list)
- б) разработана структура модифицируемой проблемной части (Record) элемента списка (возможно включение новых полей с дополнением действий

по их обработке), допускающая «произвольное» наполнение (подстраиваемый размер памяти под данные – имя и фамилию)

в) по результатам тестирования можно заключить, что обрабатываются все возможные ситуации по месту вставки очередного элемента списка (в начало, в конец, в середину)

г) реализация на C++ выполнена с использованием 2-х структур с набором соответствующих функций-членов.