МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Брянский государственный технический университет

Утверждаю
Ректор университета

_____О.Н. Федонин

«____»_____2017 г.

СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

ЛИНЕЙНЫЕ СПИСКИ

Методические указания
к выполнению лабораторной работы №3
для студентов очной, очно-заочной и заочной форм обучения
по направлениям подготовки
09.03.01 «Информатика и вычислительная техника»,
02.03.03 «Математическое обеспечение и администрирование информационных систем»,
09.03.04 «Программная инженерия»

Брянск 2017

УДК 006.91

Структуры и алгоритмы обработки данных. Линейные списки [Текст] + [Электронный ресурс]: методические указания к выполнению лабораторной работы №2 для студентов очной, очно-заочной и заочной форм обучения по направлениям подготовки 09.03.01 «Информатика и вычислительная техника», 02.03.03 «Математическое обеспечение и администрирование информационных систем», 09.03.04 «Программная инженерия»- Брянск: БГТУ, 2017. –32 с.

Разработал:
канд. техн. наук, проф.
В.К. Гулаков

Рекомендовано кафедрой «Информатика и программное обеспечение» БГТУ (протокол №1 от 13.09.17)

1. ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ

Целью лабораторной работы является:
1) освоение алгоритмов обработки линейных списков;
2) совершенствование навыков проектирования разработки, тестирования и документирования программ.

Продолжительность лабораторной работы − 4 часа.

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Однонаправленный список

Вставка элемента в однонаправленный список

В динамические структуры легко добавлять элементы, так как для этого достаточно изменить значения адресных полей. Вставка первого и последующих элементов списка отличаются друг от друга. Поэтому в функции, реализующей данную операцию, сначала осуществляется проверка, на какое место вставляется элемент. Далее реализуется соответствующий алгоритм добавления (рис.1).

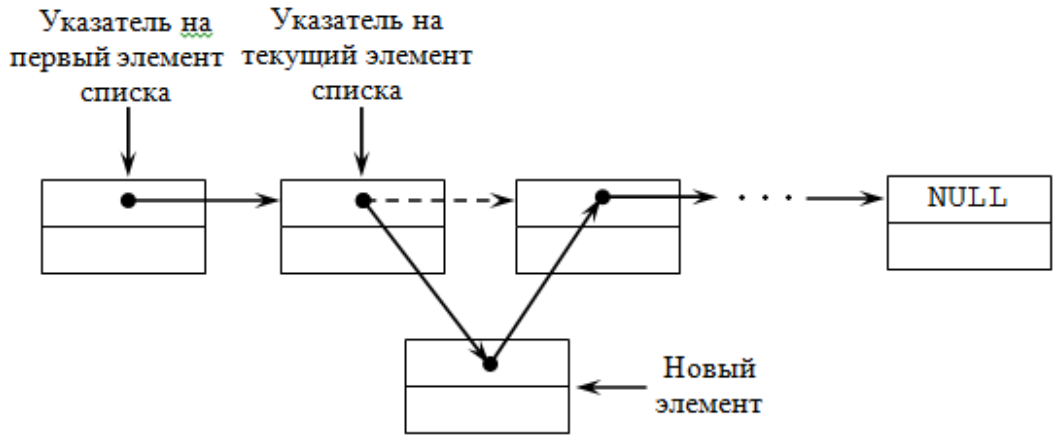

Рис. 1. Вставка элемента в однонаправленный список

```
/*вставка элемента с заданным номером в однонаправленный
список*/
Single_List* Insert_Item_Single_List(Single_List* Head,
      int Number, int DataItem){
  Number--;
  Single_List *NewItem=new(Single_List);
  NewItem->Data=DataItem;
  NewItem->Next = NULL;
  if (Head == NULL) {//список пуст
    Head = NewItem;//создаем первый элемент списка
  }
  else {//список не пуст
    Single_List *Current=Head;
    for(int i=1; i < Number && Current->Next!=NULL; i++)
    Current=Current->Next;
    if (Number == 0){
    //вставляем новый элемент на первое место
      NewItem->Next = Head;
      Head = NewItem;
    }
    else {//вставляем новый элемент на непервое место
      if (Current->Next != NULL)
            NewItem->Next = Current->Next;
      Current->Next = NewItem;
    }
  }
  return Head;
}
```

Удаление элемента из однонаправленного списка

Из динамических структур можно удалять элементы, так как для этого достаточно изменить значения адресных полей. Операция удаления элемента однонаправленного списка осуществляет удаление элемента, на который установлен указатель текущего элемента. После удаления указатель текущего элемента устанавливается на предшествующий элемент списка или на новое начало списка, если удаляется первый.

Алгоритмы удаления первого и последующих элементов списка отличаются друг от друга. Поэтому в функции, реализующей данную операцию, осуществляется проверка, какой элемент удаляется. Далее реализуется соответствующий алгоритм удаления (рис. 2.).

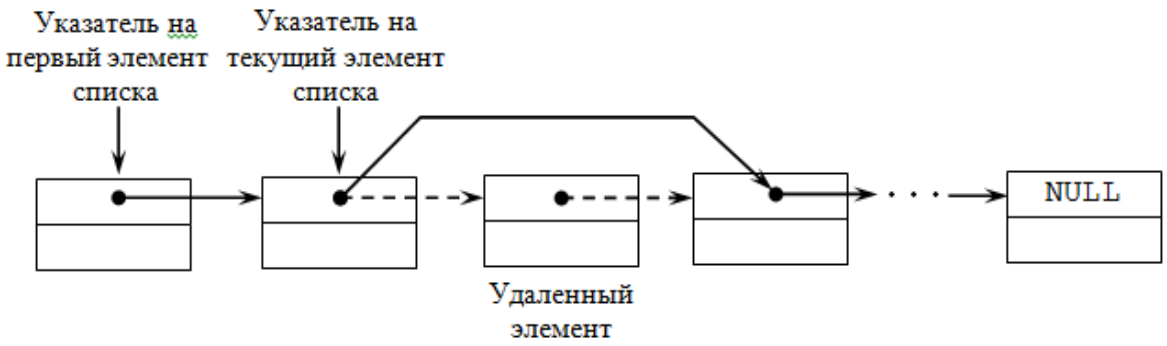

Рис. 2. Удаление элемента из однонаправленного списка

```
/*удаление элемента с заданным номером из
однонаправленного списка*/
Single_List* Delete_Item_Single_List(Single_List* Head,
      int Number){
  Single_List *ptr;//вспомогательный указатель
  Single_List *Current = Head;
  for (int i = 1; i < Number && Current != NULL; i++)
    Current = Current->Next;
  if (Current != NULL){//проверка на корректность
    if (Current == Head){//удаляем первый элемент
      Head = Head->Next;
      delete(Current);
      Current = Head;
    }
    else {//удаляем непервый элемент
      ptr = Head;
      while (ptr->Next != Current)
        ptr = ptr->Next;
      ptr->Next = Current->Next;
      delete(Current);
      Current=ptr;
    }
  }
  return Head;
}
```

Поиск элемента в однонаправленном списке

Операция поиска элемента в списке заключается в последовательном просмотре всех элементов списка до тех пор, пока текущий элемент не будет содержать заданное значение или пока не

будет достигнут конец списка. В последнем случае фиксируется отсутствие искомого элемента в списке (функция принимает значение *false*).

```
//поиск элемента в однонаправленном списке
bool Find_Item_Single_List(Single_List* Head, int
DataItem){
  Single_List *ptr; //вспомогательным указатель
  ptr = Head;
  while (ptr != NULL){//пока не конец списка
    if (DataItem == ptr->Data) return true;
    else ptr = ptr->Next;
  }
  return false;
}
```

Удаление однонаправленного списка

Операция удаления списка заключается в освобождении динамической памяти. Для данной операции организуется функция, в которой нужно переставлять указатель на следующий элемент списка до тех пор, пока указатель не станет, равен NULL, то есть не будет достигнут конец списка. Реализуем рекурсивную функцию.

```
/*освобождение памяти, выделенной под однонаправленный
список*/
void Delete_Single_List(Single_List* Head){
  if (Head != NULL){
    Delete_Single_List(Head->Next);
    delete Head;
  }
}
```

Таким образом, однонаправленный список имеет только один указатель в каждом элементе. Это позволяет минимизировать расход памяти на организацию такого списка. Одновременно это позволяет осуществлять переходы между элементами только в одном направлении, что зачастую увеличивает время, затрачиваемое на обработку списка. Например, для перехода к предыдущему элементу необходимо осуществить просмотр списка с начала до элемента, указатель которого установлен на текущий элемент.

Двунаправленные (двусвязные) списки

Для ускорения многих операций целесообразно применять переходы между элементами списка в обоих направлениях. Это реализуется с помощью двунаправленных списков, которые являются сложной динамической структурой.

Двунаправленный (двусвязный) список – структура данных, состоящая из последовательности элементов, каждый из которых содержит информационную часть и два указателя на соседние элементы (рис. 3). При этом два соседних элемента должны содержать взаимные ссылки друг на друга.

В таком списке каждый элемент (кроме первого и последнего) связан с предыдущим и следующим за ним элементами. Каждый элемент двунаправленного списка имеет два поля с указателями: одно поле содержит ссылку на следующий элемент, другое поле – ссылку на предыдущий элемент и третье поле – информационное. Наличие ссылок на следующее звено и на предыдущее позволяет двигаться по списку от каждого звена в любом направлении: от звена к концу списка или от звена к началу списка, поэтому такой список называют двунаправленным.

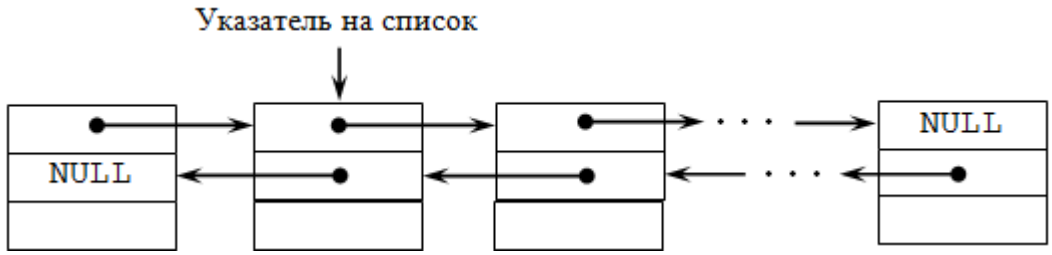

Рис. 3. Двунаправленный список

Описание простейшего элемента такого списка выглядит следующим образом:

```
struct имя_типа {
                 информационное поле;
                 адресное поле 1;
                 адресное поле 2;
                };
```

где информационное поле – поле любого, ранее объявленного или стандартного, типа;

адресное поле 1 – указатель на объект того же типа, что и определяемая структура, в него записывается адрес следующего элемента списка;

адресное поле 2 – указатель на объект того же типа, что и определяемая структура, в него записывается адрес предыдущего элемента списка, например:

```
struct list {
              type elem ;
              list *next, *pred ;
             }
list *headlist ;
```

где *type* – тип информационного поля элемента списка;

**next*, **pred* – указатели на следующий и предыдущий элементы этой структуры соответственно.

Переменная-указатель *headlist* задает список как единый программный объект, ее значение – указатель на первый (или заглавный) элемент списка.

Основные операции, выполняемые над двунаправленным списком, те же, что и для однонаправленного списка. Так как двунаправленный список более гибкий, чем однонаправленный, то при включении элемента в список, необходимо использовать указатель как на элемент, за которым происходит включение, так и указатель на элемент, перед которым происходит включение. При исключении элемента из списка необходимо использовать как указатель на сам исключаемый элемент, так и указатели на предшествующий или следующий за исключаемым элементы. Однако так как элемент двунаправленного списка имеет два указателя, то при выполнении операций включения/исключения элемента необходимо изменять больше связей, чем в однонаправленном списке.

Основными операциями, осуществляемые с двунаправленными списками являются:

- создание списка;
- печать (просмотр) списка;
- вставка элемента в список;
- удаление элемента из списка;
- поиск элемента в списке;

- проверка пустоты списка;
- удаление списка.

В отличие от однонаправленного списка здесь нет необходимости обеспечивать позиционирование какого-либо указателя именно на первый элемент списка, так как благодаря двум указателям в элементах можно получить доступ к любому элементу списка из любого другого элемента, осуществляя переходы в прямом или обратном направлении. Однако указатель желательно ставить на заголовок списка.

Для описания алгоритмов этих основных операций используется следующее объявления:

```
struct Double_List {//структура данных
                     int Data; //информационное поле
                     Double_List *Next, //адресное поле
                                 *Prior; //адресное поле
                    };
. . . . . .
Double_List *Head; //указатель на первый элемент списка
. . . . . .
Double_List *Current;
//указатель на текущий элемент списка (при необходимости)
```

Создание двунаправленного списка

Чтобы создать список, необходимо создать сначала первый элемент списка, а затем с помощью функции добавить к нему остальные элементы. Добавление может выполняться как в начало, так и в конец списка. Реализуем рекурсивную функцию.

```
//создание двунаправленного списка (добавления в конец)
void Make_Double_List(int n,Double_List** Head,
         Double_List* Prior){
  if (n > 0) {
    (*Head) = new Double_List();
     //выделяем память под новый элемент
    cout << "Введите значение ";
    cin >> (*Head)->Data;
    //вводим значение информационного поля
    (*Head)->Prior = Prior;
```

```
    (*Head)->Next=NULL;//обнуление адресного поля
    Make_Double_List(n-1,&((*Head)->Next),(*Head));
  }
  else (*Head) = NULL;
}
```

Печать (просмотр) двунаправленного списка

Операция печати списка для двунаправленного списка реализуется абсолютно аналогично соответствующей функции для однонаправленного списка. Просматривать двунаправленный список можно в обоих направлениях.

```
//печать двунаправленного списка
void Print_Double_List(Double_List* Head) {
  if (Head != NULL) {
    cout << Head->Data << "\t";
    Print_Double_List(Head->Next);
    //переход к следующему элементу
  }
  else cout << "\n";
}
```

Вставка элемента в двунаправленный список

В динамические структуры легко добавлять элементы, так как для этого достаточно изменить значения адресных полей. Операция вставки реализовывается аналогично функции вставки для однонаправленного списка, только с учетом особенностей двунаправленного списка (рис. 4).

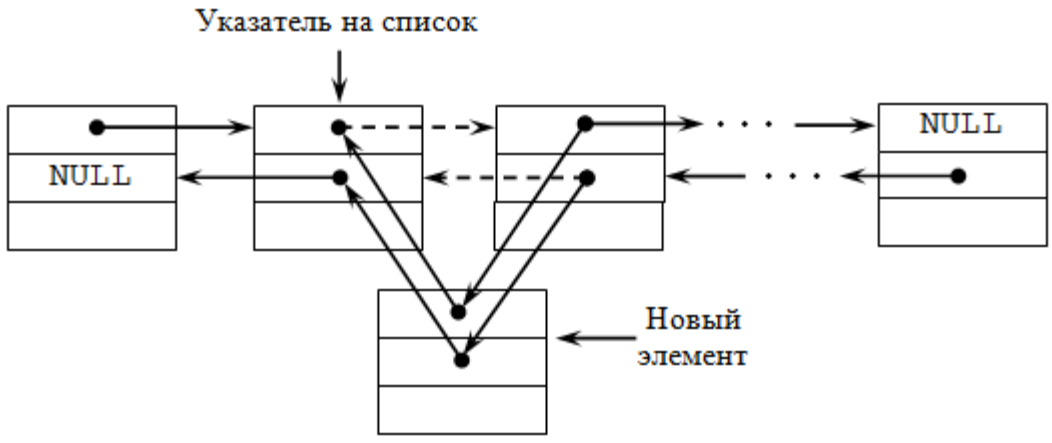

Рис. 4. Добавление элемента в двунаправленный список

```
//вставка элемента с заданным номером в двунаправленный
список
Double_List* Insert_Item_Double_List(Double_List* Head,
      int Number, int DataItem){
  Number--;
  Double_List *NewItem=new(Double_List);
  NewItem->Data=DataItem;
  NewItem->Prior=NULL;
  NewItem->Next = NULL;
  if (Head == NULL) {//список пуст
    Head = NewItem;
  }
  else {//список не пуст
    Double_List *Current=Head;
    for(int i=1; i < Number && Current->Next!=NULL; i++)
    Current=Current->Next;
    if (Number == 0){
    //вставляем новый элемент на первое место
      NewItem->Next = Head;
      Head->Prior = NewItem;
      Head = NewItem;
    }
    else {//вставляем новый элемент на непервое место
      if (Current->Next != NULL) Current->Next->Prior =
NewItem;
      NewItem->Next = Current->Next;
      Current->Next = NewItem;
           NewItem->Prior = Current;
      Current = NewItem;
    }
  }
  return Head;
}
```

Удаление элемента из двунаправленного списка

Из динамических структур можно удалять элементы, так как для этого достаточно изменить значения адресных полей. Операция удаления элемента из двунаправленного списка осуществляется во многом аналогично удалению из однонаправленного списка (рис. 5).

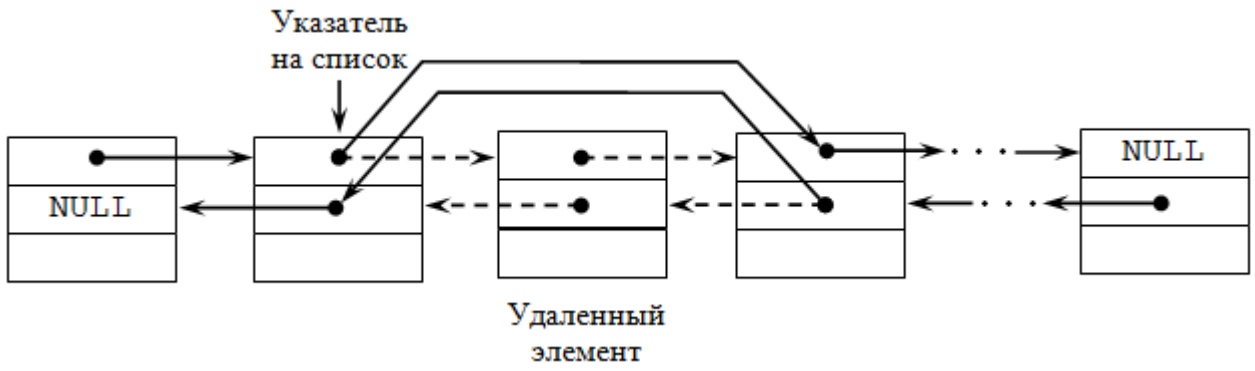

Рис. 5. Удаление элемента из двунаправленного списка

```
/*удаление элемента с заданным номером из
двунаправленного списка*/
Double_List* Delete_Item_Double_List(Double_List* Head,
      int Number){
  Double_List *ptr;//вспомогательный указатель
  Double_List *Current = Head;
  for (int i = 1; i < Number && Current != NULL; i++)
    Current = Current->Next;
  if (Current != NULL){//проверка на корректность
    if (Current->Prior == NULL){//удаляем первый элемент
      Head = Head->Next;
      delete(Current);
      Head->Prior = NULL;
      Current = Head;
    }
    else {//удаляем непервый элемент
      if (Current->Next == NULL) {
      //удаляем последний элемент
        Current->Prior->Next = NULL;
        delete(Current);
        Current = Head;
      }
      else {//удаляем непервый и непоследний элемент
        ptr = Current->Next;
        Current->Prior->Next =Current->Next;
        Current->Next->Prior =Current->Prior;
        delete(Current);
        Current = ptr;
      }
    }
  }
  return Head;
}
```

Поиск элемента в двунаправленном списке

Операция поиска элемента в двунаправленном списке реализуется абсолютно аналогично соответствующей функции для однонаправленного списка. Поиск элемента в двунаправленном списке можно вести:

а) просматривая элементы от начала к концу списка;

б) просматривая элементы от конца списка к началу;

в) просматривая список в обоих направлениях одновременно: от начала к середине списка и от конца к середине (учитывая, что элементов в списке может быть четное или нечетное количество).

```
//поиск элемента в двунаправленном списке
bool Find_Item_Double_List(Double_List* Head,
    int DataItem){
  Double_List *ptr; //вспомогательный указатель
  ptr = Head;
  while (ptr != NULL){//пока не конец списка
    if (DataItem == ptr->Data) return true;
    else ptr = ptr->Next;
    }
  return false;
}
```

Проверка пустоты двунаправленного списка

Операция проверки двунаправленного списка на пустоту осуществляется аналогично проверки однонаправленного списка.

```
//проверка пустоты двунаправленого списка
bool Empty_Double_List(Double_List* Head){
  if (Head!=NULL)  return false;
  else  return true;
}
```

Удаление двунаправленного списка

Операция удаления двунаправленного списка реализуется аналогично удалению однонаправленного списка.

```
//освобождение памяти. выделенной под двунаправленный
список
```

```
void Delete_Double_List(Double_List* Head){
  if (Head != NULL){
    Delete_Double_List(Head->Next);
    delete Head;
  }
}
```

Пример 1

N натуральных чисел являются элементами двунаправленного списка *L*, вычислить: $X_1X_n+X_2X_{n-1}+...+X_nX_1$. Вывести на экран каждое произведение и итоговую сумму.

Алгоритм:

1) создаём структуру;
2) формируем список целых чисел;
3) продвигаемся по списку: от начала к концу и от конца к началу в одном цикле, перемножаем данные, содержащиеся в соответствующих элементах списка;
4) суммируем полученные результаты;
5) выводим на печать

Создание структуры, формирование списка и вывод на печать рассмотрены ранее. Приведем функции для реализации продвижения по списку в обоих направлениях и нахождения итоговой суммы.

```
//поиск последнего элемента списка
Double_List* Find_End_Item_Double_List(Double_List*
Head){
  Double_List *ptr; //дополнительный указатель
  ptr = Head;
  while (ptr->Next != NULL){
    ptr = ptr->Next;
  }
  return ptr;
}

//итоговая сумма произведений
void Total_Sum(Double_List* Head) {
  Double_List* lel = Head;
  Double_List* mel = Find_End_Item_Double_List(Head);
  int mltp,sum=0;
  while(lel != NULL) {
    mltp = (lel->Data)*(mel->Data);//умножение элементов
    printf("\n\n%d * %d = %d",lel->Data,mel->Data,mltp);
```

```
    sum = sum + mltp;//суммирование произведений
    lel = lel->Next;
    //идем по списку из первого элемента в последний
    mel = mel->Prior;
    //идем по списку из последнего элемента в первый
  }
  printf("\n\n Итоговая сумма равна %d",sum);
}
```

Циклические (кольцевые) списки

Циклический (кольцевой) список – структура данных, представляющая собой последовательность элементов, последний элемент которой содержит указатель на первый элемент списка, а первый (в случае двунаправленного списка) – на последний.

Основная особенность такой организации состоит в том, что в этом списке нет элементов, содержащих пустые указатели, и, следовательно, нельзя выделить крайние элементы.

Циклические списки, так же как и линейные, бывают однонаправленными и двунаправленными.

Циклический однонаправленный список похож на линейный однонаправленный список, но его последний элемент содержит указатель, связывающий его с первым элементом (рис. 6).

Для полного обхода такого списка достаточно иметь указатель на произвольный элемент, а не на первый, как в линейном однонаправленном списке. Понятие «первого» элемента здесь достаточно условно и не всегда требуется. Хотя иногда бывает полезно выделить некоторый элемент как «первый» путем установки на него специального указателя. Это требуется, например, для предотвращения «зацикливания» при просмотре списка.

Основными операциями, осуществляемыми с циклическим однонаправленным списком, являются:

- создание списка;
- печать (просмотр) списка;
- вставка элемента в список;
- удаление элемента из списка;
- поиск элемента в списке;
- проверка пустоты списка;
- удаление списка.

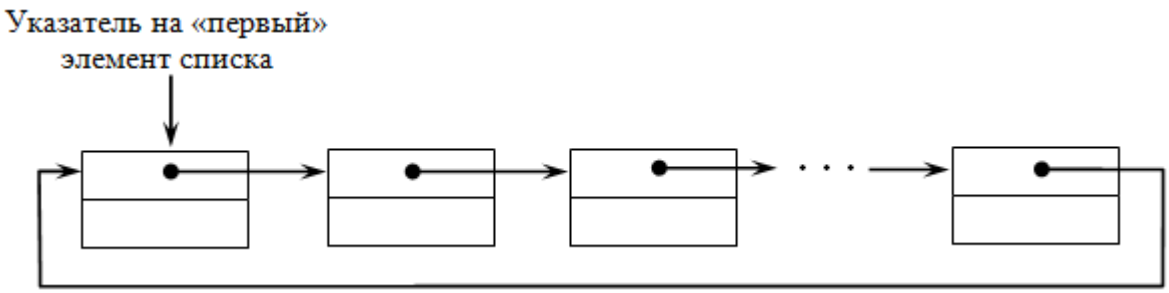

Рис. 6. Циклический однонаправленный список

Для описания алгоритмов этих основных операций будем использовать те же объявления, что и для линейного однонаправленного списка.

Приведем функции перечисленных основных операций при работе с циклическим однонаправленным списком.

```
//создание циклического однонаправленного списка
void Make_Circle_Single_List(int n,
       Circle_Single_List** Head,Circle_Single_List*
Loop){
    if (n > 0) {
         (*Head) = new Circle_Single_List();
         //выделяем память под новый элемент
         if (Loop == NULL) Loop = (*Head);
         cout << "Введите значение ";
         cin >> (*Head)->Data;
         //вводим значение информационного поля
         (*Head)->Next=NULL;//обнуление адресного поля
         Make_Circle_Single_List(n-1,&((*Head)-
>Next),Loop);
    }
    else {
         (*Head) = Loop;
    }
}

//печать циклического однонаправленного списка
void Print_Circle_Single_List(Circle_Single_List* Head) {
    Circle_Single_List* ptr=Head;
    //вспомогательный указатель
    do {
       cout << ptr->Data << "\t";
```

```
       ptr=ptr->Next;
    } while (ptr!=Head);
    cout << "\n";
}

/*вставка элемента после заданного номера в циклический
однонаправленный список*/
Circle_Single_List*
Insert_Item_Circle_Single_List(Circle_Single_List* Head,
      int Number, int DataItem){
  Circle_Single_List *Current = Head;
  //встали на первый элемент
  Circle_Single_List *NewItem = new(Circle_Single_List);
  //создали новый элемент
  NewItem->Data = DataItem;
  if (Head == NULL) {//список пуст
    NewItem->Next = NewItem;
    Head = NewItem;
  }
  else {//список не пуст
    for (int i = 1; i < Number; i++)
      Current = Current->Next;
    NewItem->Next = Current->Next;
    Current->Next = NewItem;
  }
  return Head;
}

/*удаление элемента с заданным номером из циклического
однонаправленного списка*/
Circle_Single_List* Delete_Item_Circle_Single_List
      (Circle_Single_List* Head, int Number){
  if (Head != NULL){
    Circle_Single_List *Current = Head;
    if (Head->Next != Head){
      for (int i = 1; i < Number; i++)
        Current = Current->Next;
        Circle_Single_List *ptr = Head;
      while (ptr->Next != Current)
        ptr = ptr->Next;
      //непосредственное удаление элемента
      ptr->Next = Current->Next;
      if (Head = Current) Head = Current->Next;
      delete(Current);
    }
```

```
    else{
      Head = NULL;
      delete(Current);
    }
  }
  return Head;
}

//поиск элемента в циклическом однонаправленном списке
bool Find_Item_Circle_Single_List(Circle_Single_List*
Head,
          int DataItem){
  Circle_Single_List *ptr = Head;
  //вспомогательный указатель
  do {
    if (DataItem == ptr->Data) return true;
    else ptr = ptr->Next;
    }
  while (ptr != Head);
  return false;
}

//проверка пустоты циклического однонаправленного списка
bool Empty_Circle_Single_List(Circle_Single_List* Head){
  return (Head != NULL ? false : true);
}

//удаление циклического однонаправленного списка
void Delete_Circle_Single_List(Circle_Single_List* Head){
  if (Head != NULL){
    Head = Delete_Item_Circle_Single_List(Head, 1);
    Delete_Circle_Single_List(Head);
  }
}
```

Циклический двунаправленный список похож на линейный двунаправленный список, но его любой элемент имеет два указателя, один из которых указывает на следующий элемент в списке, а второй указывает на предыдущий элемент (рис. 7).

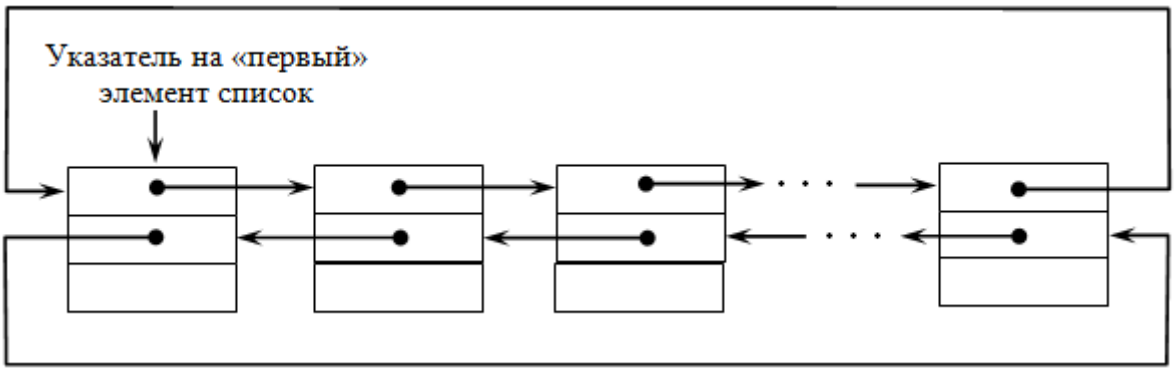

Рис. 7. Циклический двунаправленный список

Основными операциями, осуществляемыми с циклическим двунаправленным списком являются:

- создание списка;
- печать (просмотр) списка;
- вставка элемента в список;
- удаление элемента из списка;
- поиск элемента в списке
- проверка пустоты списка;
- удаление списка.

Для описания алгоритмов этих основных операций будем использовать те же объявления, что и для линейного двунаправленного списка.

Приведем функции перечисленных основных операций при работе с циклическим двунаправленным списком.

```
//создание циклического двунаправленного списка
Circle_Double_List* Make_Circle_Double_List(int n,
      Circle_Double_List** Head,Circle_Double_List*
Loop){
  Circle_Double_List* ptr;//вспомогательный указатель
  if (n > 0) {
    (*Head) = new Circle_Double_List();
    //выделяем память под новый элемент
    if (Loop == NULL) Loop = (*Head);
      cout << "Введите значение ";
    cin >> (*Head)->Data;
    //вводим значение информационного поля
    (*Head)->Next=NULL;//обнуление адресного поля
```

```
    ptr = Make_Circle_Double_List(n-1,&((*Head)-
>Next),Loop);
    if ((*Head)->Next != NULL)
      (*Head)->Next->Prior = (*Head);
    if ((*Head)->Prior == NULL)
      (*Head)->Prior = ptr;
    if (ptr == NULL)
      return *Head;
    else return ptr;
    }
    else {
      (*Head) = Loop;
    return NULL;
  }
}

//печать циклического двунаправленного списка
void Print_Circle_Double_List(Circle_Double_List* Head) {
    Circle_Double_List* ptr=Head;
    //вспомогательный указатель
    do {
       cout << ptr->Data << "\t";
       ptr=ptr->Next;
    } while (ptr!=Head);
    cout << "\n";
}

/*вставка элемента после заданного номера в циклический
двунаправленный список*/
Circle_Double_List* Insert_Item_Circle_Double_List
   (Circle_Double_List* Head, int Number, int DataItem){
  Circle_Double_List *Current = Head;
  //встали на первый элемент
  Circle_Double_List *NewItem = new(Circle_Double_List);
  //создали новый элемент
  NewItem->Data = DataItem;
  if (Head == NULL) {//список пуст
    NewItem->Next = NewItem;
    NewItem->Prior = NewItem;
    Head = NewItem;
  }
  else {//список не пуст
    for (int i = 1; i < Number; i++)
      Current = Current->Next;
    NewItem->Next = Current->Next;
```

```
    Current->Next = NewItem;
    NewItem->Prior = Current;
    NewItem->Next->Prior = NewItem;
  }
  return Head;
}

/*удаление элемента с заданным номером из циклического
двунаправленного списка*/
Circle_Double_List*
Delete_Item_Circle_Double_List(Circle_Double_List* Head,
      int Number){
  if (Head != NULL){
    Circle_Double_List *Current = Head;
    if (Head->Next != Head){
      for (int i = 1; i < Number; i++)
        Current = Current->Next;
      Circle_Double_List *ptr = Current->Next;
      Current->Prior->Next = Current->Next;
      Current->Next->Prior = Current->Prior;
      if (Head = Current) //удаляем первый
        Head = Current->Next;
      delete(Current);
    }
    else{
      Head = NULL;
      delete(Current);
    }
  }
  return Head;
}

//поиск элемента в циклическом двунаправленном списке
bool Find_Item_Circle_Double_List(Circle_Double_List*
Head,
         int DataItem){
  Circle_Double_List *ptr = Head;
  //вспомогательный указатель
  do {
    if (DataItem == ptr->Data)
      return true;
    else ptr = ptr->Next;
    }
  while (ptr != Head);
  return false;
```

```
}

//проверка пустоты циклического двунаправленного списка
bool Empty_Circle_Double_List(Circle_Double_List* Head){
  return (Head != NULL ? false : true);
}

//удаление циклического двунаправленного списка
void Delete_Circle_Double_List(Circle_Double_List* Head){
  if (Head != NULL){
    Head = Delete_Item_Circle_Double_List(Head, 1);
    Delete_Circle_Double_List(Head);
  }
}
```

Основные понятия

Двунаправленный (двусвязный) список – структура данных, состоящая из последовательности элементов, каждый из которых содержит информационную часть и два указателя на соседние элементы.

Длина списка – величина, равная числу элементов в списке.

Линейный список – список, отражающий отношения соседства между элементами.

Однонаправленный (односвязный) список – структура данных, представляющая собой последовательность элементов, в каждом из которых хранится значение и указатель на следующий элемент списка.

Пустой список – список нулевой длины.

Связанный список – структура, элементами которой являются записи одного формата, связанные друг с другом с помощью указателей, хранящихся в самих элементах.

Список – упорядоченное множество, состоящее из переменного числа элементов, к которым применимы операции включения, исключения.

Указатель начала списка (голова списка) – указатель на первый элемент списка.

Резюме

1. Список является динамической структурой, для элементов которого определены операции включения, исключения.
2. В связанном списке элементы линейно упорядочены указателями, входящими в состав элементов списка.
3. Линейные связные списки являются простейшими динамическими структурами данных и в зависимости от организации связей делятся на однонаправленные и двунаправленные.
4. В однонаправленном (односвязном) списке каждый из элементов содержит информационную часть и указатель на следующий элемент списка. Адресное поле последнего элемента имеет значение NULL.
5. Каждый элемент списка содержит ключ, который идентифицирует этот элемент.
6. Основными операциями с однонаправленными списками, являются: создание списка; печать (просмотр) списка; вставка элемента в список; удаление элемента из списка; поиск элемента в списке; проверка пустоты списка; удаление списка.
7. В двунаправленном (двусвязном) списке каждый из элементов содержит информационную часть и два указателя на соседние элементы.
8. Основные операции, выполняемые над двунаправленным списком, те же, что и для однонаправленного списка.

3. ЗАДАЧИ К ЛАБОРАТОРНОЙ РАБОТЕ

1. Сформируйте однонаправленный список с вещественным информационным полем. Добавьте в список заданный элемент после первого элемента с аналогичным информационным полем. Выполните печать списка до и после изменений. Найти среднее арифметическое элементов списка**.**

2. Сформируйте однонаправленный список с информационным полем типа *char**. Добавьте в этот список элементы с нечетными номерами. Решите задачу, выполняя следующие требования:

- Сформировать однонаправленный список, тип информационного поля указан в варианте.
- Распечатать полученный список.

- Выполнить обработку списка в соответствии с заданием.
- Распечатать полученный список.
- Удалить список из памяти.

3. Для решения задачи сформируйте двунаправленный список с символьным информационным полем. Дана последовательность латинских букв, оканчивающаяся точкой. Среди букв есть специальный символ *Ch*, появление которого означает отмену предыдущего символа. Учитывая вхождение этого символа, преобразуйте последовательность.

4. Имеются N линейных списков, имеющих одинаковую структуру и упорядоченных по возрастанию некоторого ключевого поля. Создайте общий упорядоченный список.

5. В общежитии живет N студентов. При поселении каждый студент представил список своих знакомых. Известно, что знакомые любого студента вскоре знакомятся между собой. Выясните, будут ли все студенты знакомы между собой.

6. Напишите алгоритм и программу следующих операций:

− добавление элемента к концу списка;

− сцепление двух списков;

− освобождение всех элементов в списке;

− инвертирование списка (первый элемент становится последним и т.д.);

− удаление последнего элемента из списка;

− удаление n-го элемента из списка;

− объединение двух упорядоченных списков в один упорядоченный список.

7. Напишите алгоритм и программу следующих операций:

− создание списка, содержащего элементы, общие для двух других списков;

− вставка элемента после n-го элемента списка;

− удаление из списка каждого второго элемента;

− размещение элементов списка в возрастающем порядке;

− вычисление суммы целочисленных значений элементов списка;

− вычисление числа элементов в списке;

− перемещение произвольного элемента списка на n позиций вперед.

8. Напишите алгоритм и программу, меняющую местами *m*-й и *n*-й элементы списка.

9. Связи в связанном списке содержат ключевое поле типа *integer*. Напишите программу сортировки списка в порядке возрастания значения ключей. Затем сформулируйте процедуру, формирующую список, в котором элементы расположены в обратном порядке.

10. В круговых списках обычно предусматривается так называемый заголовок списка. Зачем необходимы такие заголовки? Напишите процедуры для включения, исключения и поиска элемента, идентифицированного заданным ключом. Проделайте это для списка с заголовком, а потом − без заголовка.

11. Напишите программы вставки, удаления и поиска элементов отсортированного списка, используя для реализации списка

а) массив;

б) указатели;

в) курсоры.

Каково время выполнения каждой из этих программ?

12. Напишите программу для слияния

а) двух отсортированных списков;

б) *n* отсортированных списков.

13. Многочлены вида $p(x) = c_1x^{e1} + c_2x^{e2} + \ldots + c_nx^{en}$, где $e_1>e_2>...>e_n>0$, можно представить в виде связанного списка, где каждая запись имеет три поля: одно – для коэффициента c_i второе – для показателя степени e_i, третье – для указателя на следующую запись. Для описанного представления многочленов напишите программу их дифференцирования.

14. Напишите программу сложения и умножения многочленов, используя их представление, описанное в задании 32.

15. Следующая процедура предназначена для удаления всех вхождений элемента *x* в списке *L*. Найдите причину, по которой эта процедура будет выполняться не всегда, и устраните ее.

```
    procedure delete ( x: elementtype; var
L: LIST );
    var
    p:position;
    begin
```

```
            p: FIRST(L);
            while p < > END(L) do begin
            if RETREVE(p,L) = x then
            DELETE(p,L);
            P: =NEXT(p,L)
            end
end; ( delete }
```

16. Сохраните список в массиве *A*, чьи записи содержат два поля: *data* – для элементов и поле *position* – для позиций (целых чисел) элементов. Целочисленная переменная *last* (последний) используется для указания того, что список содержится в записях от *A*[1] до *A[last]* массива *A*. Тип *LIST* определен следующим образом:

```
    type
     LIST = record
                              last:
 integer;
                              elements: array[1. .
maxlength] of record
                                 data: elementtype;
                                  position: integer;
                              end
     end;
```

Напишите процедуру *DELETE*(*p, L)* для удаления элемента в позиции *p.* Включите в процедуру все необходимые проверки на «внештатные» ситуации.

17. Пусть *L* — это список типа *LIST*, *p, q* и *r* — позиции. Определите как функцию от n (длины списка *L)* количество выполнений функций *FIRST*, *END* и *NEXT* в следующей программе

```
p:= FIRST (L);
while p < > END(L) do begin
   q:= p;
   while q < >END(L) do begin
   q:= NEXT(q, L);
   r:= FIRST (L);
     while r < > q do
              r:= NEXT(r, L)
   end;
   p:= NEXT(p, L)
end;
```

18. Выполнить сложение длинных положительных целых чисел при помощи циклических списков.

Указание. Для сложения двух чисел их цифры просматриваются справа налево и соответствующие пары складываются друг с другом с учетом переноса, полученного при сложении предыдущей пары. Это предполагает хранение длинных целых чисел в списке с размещением их цифр справа налево так, что первый элемент списка содержит последнюю значащую цифру (крайнюю правую), а последний элемент содержит первую значащую цифру (крайнюю левую). Однако для экономии пространства лучше хранить, например, по пять цифр в каждом элементе. Поскольку в процессе сложения необходимо просматривать список и при этом сохранять исходные значения указателей списка, то следует воспользоваться циклическим списком с заголовком.

19. Напишите программу, которая удаляет узлы связного списка, находящиеся в позициях с номерами, кратными 5.

20. Напишите программу, которая перемещает наибольший элемент данного списка в конец списка.

21. Напишите программу, которая меняет местами два заданных элемента двухсвязного списка.

22. Напишите программу, которая определяет количество узлов в циклическом списке, находящихся между узлами, на которые ссылаются два заданных указателя *x* и *t*.

23. Имеется словарь, реализованный с помощью списка. Каждый элемент списка состоит из корня слова, указателя на список окончаний и указателя на следующий элемент списка. Необходимо написать программу, позволяющую реализовать операции:

поиска заданного слова;

вставки заданного слова;

удаления заданного слова.

24. Имеется бесконечно большое число, представить его в виде списка цифр, Реализуйте в виде меню операции сложения, вычитания, умножения и деления.

25. Напишите программу объединения списков в один список.

26. Напишите процедуру обмена элементами в позициях p и NEXT(p) для простого связанного списка.

В упражнениях 16.14—16.29 *использовать (линейные) однонаправленные списки без заглавного звена* (рис. 12,а) *или с заглавным звеном* (рис. 12,б) *при следующем их описании:*

```
type ТЭ = ...; {тип элементов списка (уточняемый,
        если надо, в упражнениях)}
список = t звено;
звено = record элем:ТЭ; след:список end;
```

При этом параметры L, L1 и L2 обозначают списки, а параметры E, E1 и E2— данные типа ТЭ, к которым применимы операции присваивания и проверки на равенство.

27. Описать функцию или процедуру, которая:

а) определяет, является ли список *L* пустым;

б) находит среднее арифметическое элементов непустого списка *L (ТЭ—real);'*

в)* заменяет в списке *L* все вхождения *E1* на *E2*;

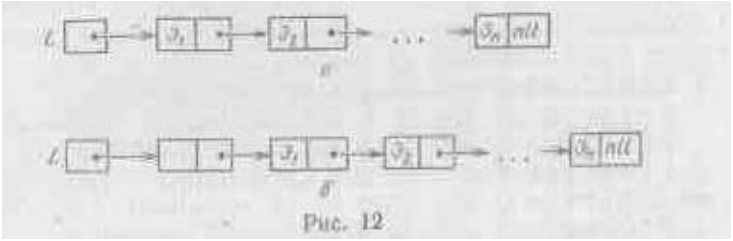

Рис. 12

г) меняет местами первый и последний элементы непустого списка L;

д)* проверяет, упорядочены ли элементы списка *L* по алфавиту *(ТЭ = 'a'..'z');*

е) находит сумму последнего и предпоследнего элементов списка L, содержащего не менее двух элементов

(ТЭ=integer).

28. type слово=packed array [1..10] of char;
ТЭ=слово;

Описать функцию, подсчитывающую количество слов списка L, которые!

а) начинаются и оканчиваются одной и той же литерой;

б) начинаются с той же литеры, что и следующее слово;

в) совпадают с последним словом.

29. type файл=file of ТЭ;
массив = аггау [1..50] of ТЭ;

Описать функцию, значением которой является список, построенный из элементов:

а)* файла *f*;

б) массива *x* (список строить от конца).

30. Описать процедуру, которая по списку *L* строит два новых списка: *L1* - из положительных элементов и *L2* - из остальных элементов списка

L (ТЭ=гea1).

31. Описать процедуру, которая вставляет:

а) в начало списка *L* новый элемент *E*;

б) в конец списка *L* новый элемент *E*;

в) новый элемент *E* после первого элемента непустого списка *L*;

г) в список *L* новый элемент *E1* за каждым вхождением элемента *E*;

д)* в список *L* новый элемент *E1* перед первым вхождением элемента *E,* если *E* входит в *L*;

е) в непустой список *L* пару новых элементов *E1* и *E2* перед его последним элементом;

ж) в непустой список *L*, элементы которого упорядочены по неубыванию, новый элемент *E* так, чтобы сохранилась упорядоченность *(ТЭ=real).*

32. Описать процедуру, которая удаляет:

а) из непустого списка *L* первый элемент;

б) из списка *L* второй элемент, если такой есть;

в) из списка *L* за каждым вхождением элемента *E* один элемент, если такой есть и он отличен от *E*;

г)* из непустого списка *L* последний элемент;

д) из списка *L* первый отрицательный элемент, если такой есть *(ТЭ=integer);*

е) из списка *L* все отрицательные элементы *(ТЭ=rea1).*

33*. Программа. Заданный во входном файле текст (за ним следует точка) распечатать в обратном порядке.

34. Программа. Дана непустая последовательность натуральных чисел, за которой следует 0. Напечатать порядковые номера тех чисел последовательности, которые имеют наибольшую величину.

35. Программа. Дано целое *n>1*, за которым следует *n* вещественных чисел. Напечатать эти числа в порядке их неубывания.

36. Описать процедуру или функцию, которая:

а) проверяет на равенство списки *L1* и *L2*;

б) определяет, входит ли список *L1* в список *L2*;

в) проверяет, есть ли в списке *L* хотя бы два одинаковых элемента;

г) переносит в конец непустого списка *L* его первый элемент;

д) переносит в начало непустого списка *L* его последний элемент;

е)* добавляет в конец списка *L1* все элементы списка *L2*;

ж) вставляет в список *L* за первым вхождением элемента *E* все элементы списка *L1*, если E входит в *L;*

з) переворачивает список *L,* т. е. изменяет ссылки в этом списке так, чтобы его элементы оказались расположенными в обратном порядке;

и) в списке *L* из каждой группы подряд идущих равных элементов оставляет только один;

к) оставляет в списке *L* только первые вхождения одинаковых элементов.

37. Описать рекурсивную функцию или процедуру, которая:

а)* определяет, входит ли элемент *E* в список *L*;

б) подсчитывает число вхождений элемента *E* в список *L*;

в) находит максимальный элемент непустого списка *L (ТЭ=rea1);*

г) печатает в обратном порядке элементы списка *L (ТЭ=char);*

д) заменяет в списке *L* все вхождения *E1* на *E2*;

е) * удаляет из списка *L* первое вхождение элемента *E*, если такое есть;

ж) удаляет из списка *L* все вхождения элемента *E;*

з) строит *L1* - копию списка *L;*

и) удваивает вхождение элемента *E* в список *L*;

к) находит среднее арифметическое всех элементов списка *L (ТЭ=real);*

38. Описать процедуру, которая формирует список *L*, включив в него по одному разу элементы, которые:

а) входят хотя бы в один из списков *L1* и *L2;*

б) входят одновременно в оба списка *L1* и *L2;*

в) входят в список *L1*, но не входят в список *L2;*

г) входят в один из списков *L1* и *L2*, но в тоже время не входят в другой из них.

39. Описать процедуру, которая объединяет два упорядоченных по неубыванию списка *L1* и *L2 (ТЭ=real)* в один упорядоченный по неубыванию список:

а) постороив новый список *L*;

б) меняя соответствующим образом ссылки в *L1* и *L2* и присвоив полученный список параметру *L1*.

40. Описать процедуру *подстановка (L,L1,L2)*, которая в списке L заменит первое вхождение списка *L1* (если такое есть) на список *L2*.

41. const n=///; {целая константа>1}

tupe число=packed array [l…n] of 0…9;

ТЭ=число;

Описать процедуру *упор(L),* упорядочивающую по неубыванию числа непустого списка *L* с помощью следующего алгоритма (рис. 13, где *n* предполагается равным 2).

Создать 10 пустых подсписков (по количеству цифр), а затем, просматривая числа исходного списка, занести в *k-й* подсписок все числа, оканчивающиеся цифрой *&*, после чего эти подсписки объединить в один список *L*,

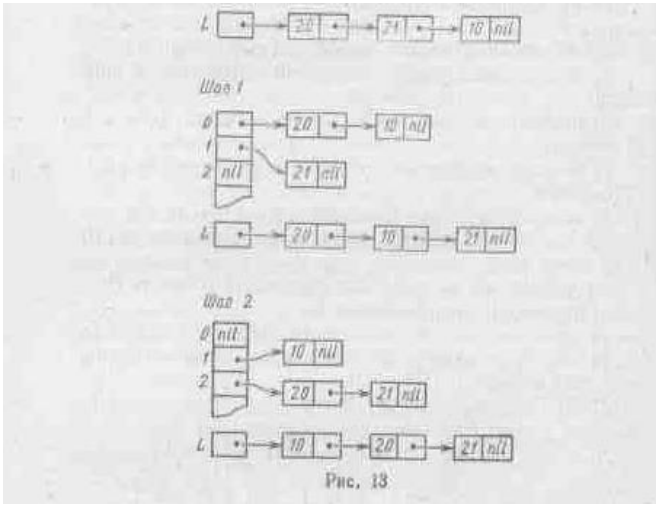

Рис. 13

записав в последнее звено *k-го* подсписка ссылку на начало (*k*+1)-го подсписка. Далее аналогичный метод применяется по отношению к предпоследней цифре чисел (не нарушая при этом упорядоченность по последней цифре), затем—по отношению к третьей от конца цифре и т. д.

42. type слово—↑цепочка;

цепочка=record буква: 'a'…'z';

связать слово end;

ТЭ=слово;

Описать функцию или процедуру, которая:

а) в списке *L* переставляет местами первое и последнее непустые слова, если в *L* есть хотя бы два непустых слова;

б)* печатает текст из первых букв всех непустых слов списка *L*;

в) удаляет из непустых слов списка *L* их первые буквы:

г) печатает все непустые слова списка *L*;

д) определяет количество слов в непустом списке *L*, отличных от последнего.

43. Многочлен

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

с целыми коэффициентами можно представить в виде списка (рис. 14, а), причем если $a_i=0$, то соответствующее звено

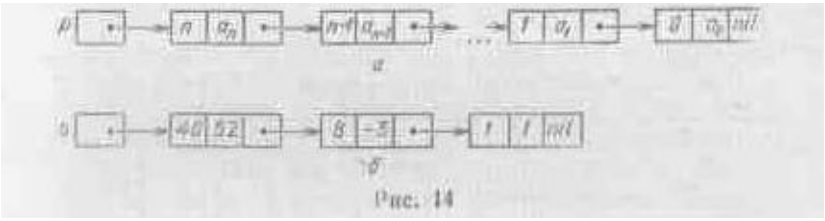

Рис. 14

не включается в список (на рис. 14, б показано представление многочлена $S(x)=52x^{40}—3x^8+x$).

Описать на Паскале тип данных, соответствующий такому представлению многочленов, и определить следующие функции и процедуры для работы с этими списками-многочленами:

а) логическую функцию *равно(p,q),* проверяющую на равенство многочлены *p* и *q;*

б) функцию *знач(p,x),* вычисляющую значение многочлена *p* в целочисленной точке *x;*

в) процедуру *диф(p,q),* которая строит многочлен *p*— производную многочлена *q;*

г) процедуру *слож(p,q,r),* которая строит многочлен *p*— сумму многочленов *q* и *r;*

д) процедуру *вывод(p,v),* которая печатает многочлен *p* как многочлен от переменной, однобуквенное имя которой является значением литерного параметра *v*; например, для указанного выше многочлена S процедура *вывод(s,'y')* должна напечатать

52y↑40-3y↑8+y;

е) процедуру *ввод(p),* которая считывает из входного файла безошибочную запись многочлена (за ней—пробел) и формирует соответствующий список-многочлен *p*.

44. Предложить и описать на Паскале представление файлов (из элементов некоторого типа *ТЭ*) в виде списков и определить функцию *eofl(f)* и процедуры *reset1(f), readl(f,x), rewritel(f)* и *writel(f,x),* реализующие при таком представлении файлов действия соответствующих стандартных функции и процедур.

45. Назовем (иерархическим) «списком» заключенную в круглые скобки последовательность элементов, разделенных запятыми; элементы списка—это атомы или снова списки :

<список>::= () | (<элементы>)

<элементы>:: = <элемент> | <элемент>,<элементы>

<элемент>::=<атом> | <список>

Под «атомом» понимается последовательность, содержащая от 1 до *n* букв и цифр, где *n*—заранее известное натуральное число. Пример подобного списка: (AD75,(3,(),(7H))). Предложить и описать на Паскале представление таких списков и определить следующие рекурсивные функции и процедуры для работы с ними:

а) логическую функцию *member(A,L),* проверяющую, входит ли атом *A в* список *L*;

б) логическую функцию *equal(L1,L2),* проверяющую на равенство списки *L1* и *L2*;

в) процедуру *printat(L),* печатающую все атомы, входящие в список *L*;

г) процедуру *printlist(L}t* печатающую список *L* в том виде, как он определен указанными выше металингвистическими формулами;

д) процедуру *readlist(L),* которая считывает из входного файла записанный без ошибок список и строит *L*— соответствующее представление этого списка.

46. Пусть *L* обозначает кольцевой (циклический) двунаправленный список с заглавным звеном (рис. 15) при следующем описании такого списка:

```
type ТЭ2—...; {тип элементов списка}
     список2=↑звено2;
     звено2=record элем:ТЭ2;
                   пред,след:список2 end;
```

и пусть *E* обозначает величину типа *ТЭ2.*

Описать функцию или процедуру, которая:

а)* определяет, является ли список *L* пустым;

б)* печатает в обратном порядке элементы непустого списка *L (ТЭ2=-char);* .

в) подсчитывает количество элементов списка *L*, у которых равные «соседи»;

г) определяет, есть ли в списке *L* хотя бы один элемент, который равен следующему за ним (по кругу) элементу;

д) в списке *L* переставляет в обратном порядке все элементы между первым и последним вхождениями элемента *E,* если *E* входит в L не менее двух раз;

е) удаляет из списка *L* первый отрицательный элемент, если такой есть;

ж) из списка L, содержащего не менее двух элементов, удаляет все элементы, у которых одинаковые «соседи» (первый и последний элементы считать соседями);

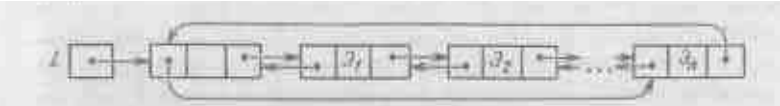

Рис. 15

з) добавляет в конец списка *L* новый элемент *E*;

и) в списке *L* удваивает каждое вхождение элемента *E*;

к) строит список *L* по однонаправленному списку *L1*;

л) в конец непустого списка *L* добавляет все его элементы, располагая их в обратном порядке (например, по списку из элементов 1, 2, 3 требуется построить список из элементов 1, 2, 3, 3, 2, 1);

47. («Считалка».) *n* ребят располагаются по кругу. Начав отсчет от первого, удаляют каждого *k-го,* смыкая круг после каждого удаления. Определить порядок удаления ребят из круга.

Решение этой задачи описать в виде программы (ее исходные данные— натуральные числа *n* и *k*), которая должна напечатать номера ребят в том порядке, как они удаляются из круга.

Решения задач 16.35—16.46 *описать в виде программ, выбрав для представления данных подходящую списковую структуру.*

48. Определить, симметричен ли заданный во входном файле текст (за ним следует точка).

49. Дана последовательность из не менее чем двух различных натуральных чисел, за которой следует 0. Напечатать в обратном порядке все числа между наибольшим и наименьшим числами этой последовательности.

50. Дана непустая последовательность непустых слов из букв; между соседними словами—запятая, за последним словом—точка. Напечатать все слова максимальной длины.

51. Дана непустая последовательность слов, в каждом из которых от 1 до 12 строчных латинских букв; между словами — пробел, за последним словом — точка. Напечатать эти слова по алфавиту, указав для каждого из них число его вхождений в эту последовательность.

52. Решить предыдущую задачу в предположении, что максимальная длина слов заранее не известна.

53. Дана непустая последовательность слов, в каждом из которых от 1 до 8 строчных латинских букв; между словами — пробел, за последним словом — точка. Напечатать эти слова в следующем порядке; сначала — по алфавиту все слова из одной буквы, затем — по алфавиту все двухбуквенные слова и т. д. (одинаковые слова печатать по одному разу).

54. Решить предыдущую задачу в предположении, что максимальная длина слов заранее не известна.

55. Дан текст, оканчивающийся точкой. Среди литер этого текста особую роль играет знак #, появление которого в тексте означает отмену предыдущей литеры текста; *k* знаков # подряд отменяют *k* предыдущих литер (если такие есть). Напечатать данный текст, исправленный с учетом такой роли знака # (например, текст ХЭ#Е##HELO#LO должен быть напечатан в виде HELLO).

56. Дано произвольное натуральное число *n*. Напечатать все цифры десятичной записи числа *n*!

57. Дано целое $n>2$. Напечатать коэффициенты *k*-го многочлена Чебышева $T_k(x)$, определяемого формулами

$$T_0(x)=1;\ T_1(x)=x;\ T_k(x)=2xT_{k-1}(x)-T_{k-2}(x) \quad (k=2, 3, \ldots).$$

58. Дана запись многочлена (от переменной x) произвольной степени с целыми коэффициентами, причем его одночлены могут быть и не упорядочены по степеням x, а одночлены одной и той же степени могут повторяться.

Возможный пример:

-8x↑4-74x+8x↑4+5-x↑3.

Требуется привести подобные члены в этом многочлене, после чего распечатать его по убыванию степеней x.

4. УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ

Каждую задачу решить в соответствии с изученными методами формирования, вывода и обработки данных однонаправленных или

двунаправленных списков в языке C++. Обработку списков выполнить на основе базовых алгоритмов: поиск по списку, вставка элемента в список, удаление элемента из списка, удаление всего списка. При объявлении списков выполните комментирование используемых полей. Программу для решения каждой задачи разработать методом процедурной абстракции, оформив комментарии к коду.

Каждую задачу реализовать в соответствии с приведенными этапами:

- изучить словесную постановку задачи, выделив при этом все виды данных;
- сформулировать математическую постановку задачи;
- выбрать метод решения задачи, если это необходимо;
- разработать графическую схему алгоритма;
- записать разработанный алгоритм на языке C++;
- разработать контрольный тест к программе;
- отладить программу;
- составить отчет о лабораторной работе.

5. ТРЕБОВАНИЯ К ОТЧЕТУ

В отчете о лабораторной работе должны содержаться:

- Титульный лист.
- Словесная постановка задачи. В этом подразделе проводится полное описание задачи. Описывается суть задачи, анализ входящих в нее физических величин, область допустимых значений, единицы физических величин, возможные ограничения, анализ условий при которых задача имеет решение (не имеет решения), анализ ожидаемых результатов.
- Математическая модель. В этом подразделе вводятся математические описания физических величин и математическое описание их взаимодействий. Цель подраздела – представить решаемую задачу в математической формулировке.
- Алгоритм решения задачи. В подразделе описывается разработка структуры алгоритма, обосновывается абстракция данных, задача разбивается на подзадачи. Схема алгоритма выполняется по ЕСПД (ГОСТ 19.003-80 и ГОСТ 19.002-80).

- Листинг программы. Подраздел должен содержать текст программы на языке программирования C++, реализованный в среде MS Visual Studio 2010.
- Контрольный тест. Подраздел содержит наборы исходных данных и полученные в ходе выполнения программы результаты.
- Выводы по лабораторной работе.
- Ответы на контрольные вопросы.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Любой ли список является связным? Обоснуйте ответ.
2. В чем заключается отличие первого элемента однонаправленного (двунаправленного) списка от остальных элементов этого же списка?
3. В чем заключается отличие последнего элемента однонаправленного (двунаправленного) списка от остальных элементов этого же списка?
4. Почему при работе с однонаправленным списком необходимо позиционирование на первый элемент списка?
5. Почему при работе с двунаправленным списком необязательно позиционирование на первый элемент списка?
6. В чем заключается принципиальные отличия выполнения добавления (удаления) элемента на первую и любую другую позиции в однонаправленном списке?
7. В чем заключается принципиальные отличия выполнения основных операций в однонаправленных и двунаправленных списках?
8. С какой целью в программах выполняется проверка на пустоту однонаправленного (двунаправленного) списка?
9. С какой целью в программах выполняется удаление однонаправленного (двунаправленного) списка по окончании работы с ним? Как изменится работа программы, если операцию удаления списка не выполнять?

7. СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Кнут, Д. Искусство программирования для ЭВМ Т.1,

Основные алгоритмы / Д. Кнут – М.: Вильямс 2000. – 215 с.

2. Вирт, Н. Алгоритмы и структуры данных / Н. Вирт – М.: МИР, 1989. – 363 с.

3. Кнут, Д. Искусство программирования для ЭВМ Т. 3. Сортировка и поиск / Д. Кнут – М: Вильямс, 2000. – 245 с.

4. Пападимитриу, Х. Комбинаторная оптимизация. Алгоритмы и сложность: пер. с англ. / Х. Пападимитриу, К.М. Стайглиц. – М.: Мир 1985. – 512 с.

5. Топп, У. Структуры данных в С ++: пер. с англ. / У. Топп, У. Форд. – М.: БИНОМ, 1999. – 816 с.

6. Кормен, Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. – М: МЦНМО, 1999. –960 с.

7. Гэри, М. Вычислительные машины и труднорешаемые задачи / М. Гэри, Д. Джонсон. – М.: Мир, 1982. – 416 с.

8. Ахо, А. В. Структуры данных и алгоритмы / А. В. Ахо, Д. Э. Хопкрофт, Д. Д. Ульман. – М.: Вильямс, 2000. – 384 с.

9. Кубенский, А. А. Создание и обработка структур данных в примерах на Java / А. А.Кубенский. – СПб.: БХВ-Петербург, 2001. – 336 с.

10. Седжвик, Р. Фундаментальные алгоритмы на С++. Анализ. Структуры данных. Сортировка. Поиск: пер. с англ. /Р. Седжвик. – К.: ДиаСофт, 2001. – 688 с.

11. Хэзфилд, Р. Искусство программирования на С. Фундаментальные алгоритмы, структуры данных и примеры приложений. Энциклопедия программиста: пер. с англ. / Р. Хэзфилд, Л. Кирби [и др.]. – К.: ДиаСофт, 2001. – 736 с.

12. Мейн, М. Структуры данных и другие объекты в С++: пер. с англ. / М .Мейн, У. Савитч. – 2-е изд.– М.: Вильямс, 2002. – 832 с.

13. Хусаинов, Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си (+ CD): учеб. Пособие / Б.С. Хусаинов. – М.: Финансы и статистика, 2004. – 464 с.

14. Макконнелл, Дж. Основы современных алгоритмов / Дж. Макконнелл. – 2-е изд. – М.: Техносфера, 2004. – 368с.

15. Гудман, С.Введение в разработку и анализ алгоритмов / С. Гудман, С.Хидетнием. – М.: Мир, 1981. – 206 с.

16. Ахо, А. Построение и анализ вычислительных алгоритмов / А. Ахо, Дж.Хопкрофт, Дж.Ульман. – М.: МИР, 1979. – 329 с.

17. Сибуя, М. Алгоритмы обработки данных / М. Сибуя, Т. Яматото. – М.:МИР, 1986. – 473 с.

18. Лэнгсам, Й.Структуры данных для персональных ЭВМ / Й. Лэнгсам, М. Огенстайн, А. Тененбаум. – М.: МИР, 1989. – 327 с.

19. Гулаков, В.К. Деревья: алгоритмы и программы / В.К. Гулаков. – М.: Машиностроение-1, 2005. – 206 с.

20. Гулаков, В.К. Многомерные структуры данных / В.К. Гулаков, А.О. Трубаков. – Брянск: БГТУ, 2010. – 387 с.

Структуры и алгоритмы обработки данных. Линейные списки: методические указания к выполнению лабораторной работы №3 для студентов очной, очно-заочной и заочной форм обучения по направлениям подготовки 230100 «Информатика и вычислительная техника», 010500 «Математическое обеспечение и администрирование информационных систем», 231000 «Программная инженерия»

ВАСИЛИЙ КОНСТАНТИНОВИЧ ГУЛАКОВ

Научный редактор В.В. Конкин
Редактор издательства Л.Н. Мажугина
Компьютерный набор В.К. Гулаков

Темплан 2013 г., п.217

Подписано в печать Формат 1/16 Бумага офсетная. Офсетная печать. Усл.печ.л. 1,86 Уч.-изд.л. 1,86 Тираж 40 экз. Заказ Бесплатно

Издательство Брянского государственного технического университета
241035, Брянск, бульвар им.50-летия Октября, 7, БГТУ, тел. 58-82-49
Лаборатория оперативной полиграфии БГТУ, ул. Институтская, 16