Санкт-Петербургский государственный политехнический университет
Кафедра “Прикладная математика”

Отчет по лабораторной лаботе 2

“Алгоритмы и сруктуры данных”

Студент группы № 5030102/20002

ФИО Двас Павел Григорьевич

Выполнил (дата) 31.10.2023

Оглавление

Постановка задачи

Написать программу, осуществляющую работу с базой данных «Записная книжка». Элемент данных - фамилия и телефон. Ключ для поиска – фамилия. Базу данных хранить в памяти в виде массива самоорганизующихся списков проиндексированного буквами алфавита. Добавление выполнять в начало соответствующего списка. Написать процедуры поиска, удаления и сортировки заданного списка. Базу данных зачитывать и сохранять в файл.

Описание алгоритма

Для описания алгоритма требуется пояснить как работает двухсвязный список.

Двухсвязный список – структура данных, в которой каждый элемент (узел) хранит информацию, а также ссылку на следующий элемент и на предыдущий. Последний элемент списка ссылается «вперед» на NULL, первый – «назад» на NULL.

Для нас двухсвязный список полезен тем, что:

1. Он очень просто устроен и все алгоритмы интуитивно понятны
2. Двухсвязный список – хорошее упражнение для работы с указателями
3. Его очень просто визуализировать, это позволяет "в картинках" объяснить алгоритм

Двухсвязный список состоит из узлов. Каждый узел содержит значение и указатель на следующий и предыдущие узлы, поэтому представим его в качестве структуры

```
typedef struct tagCONTACT CONTACT;

struct tagCONTACT
{
  char SurName[MAX_NAME_SIZE];
  char PhoneNum[PHONE_LENGTH];
  CONTACT *Next, *Prev;
};
```

Где char *SurName – фамилия человека, char *PhoneNum – его номер телефона, CONTACT *Next, *Prev – указатели на следующий и предыдущий элементы списка

Помимо этого используем массив наших списков

```
typedef struct LIST
{
  char Letter;

  CONTACT *Head;
} LIST;

LIST Base[26];
```

В нём 26 элементов по количеству букв в алфавите.

Добавление элемента

Для добавления нового узла необходимо:

1. Выделить под него память
2. Задать ему значение
3. Сделать так, чтобы он ссылался на предыдущий и на следующий элементы (или на NULL, если его не было)
4. Перекинуть указатель head на новый узел.

Поиск элемента

Для поиска элемента нужно:

1. Сделать цикл по всему списку
2. Сравнить элемент списка с искомым элементом
3. Если этот элемент не равен, переходим к шагу переписываем структуру на структуру со следующим указателем. Иначе, записываем нужные нам данные и процедура завершается.
4. Если элемент не был найден передаем в переменные пустые значения.

Сортировка

Сортировку реализуем методом selection sort

1. На каждой итерации ищем наименьший элемент в неотсортированной части массива.
2. Меняем его местами с первым элементом в неотсортированной части массива.

Текст программы

List2.h:

```
/* Dvas Pavel, 5030102/20002 */

#define MAX_NAME_SIZE 100
#define PHONE_LENGTH 13

typedef struct tagCONTACT CONTACT;

struct tagCONTACT
{
  char SurName[MAX_NAME_SIZE];
  char PhoneNum[PHONE_LENGTH];
  CONTACT *Next, *Prev;
};

typedef struct LIST
{
  char Letter;

  CONTACT *Head;
} LIST;

void InitList( void );

int AddToList( char *NewSurName, char *NewPhoneNum );
void DelFromList( char *OldSurName, char *OldPhoneNum );

char * FindContact( char *SurName );

void SortList( char C );

int SaveDB( char *FileName );
int LoadDB( char *FileName );

void ClearList( void );
```

list2.c:

```
/* Dvas Pavel, 5030102/20002 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "list2.h"

LIST Base[26];

void InitList( void )
{
  int i;

  for (i = 0; i < 26; i++)
    Base[i].Letter = 'A' + i;
}

int AddToList( char *NewSurName, char *NewPhoneNum )
{
  CONTACT *NewElement;
  int ind;

  if ((NewElement = malloc(sizeof(CONTACT))) == NULL)
    return 0;

  strcpy(NewElement->PhoneNum, NewPhoneNum);
  strcpy(NewElement->SurName, NewSurName);
```

```
  ind = NewElement->SurName[0] - 'A';

  if (ind < 0 || ind > 25)
    return 0;

  NewElement->Next = Base[ind].Head;
  if (Base[ind].Head != NULL)
    Base[ind].Head->Prev = NewElement;
  NewElement->Prev = NULL;
  Base[ind].Head = NewElement;
  return 1;
}

int AddToListBack( char *NewSurName, char *NewPhoneNum )
{
  CONTACT *NewElement, *OldFirst, **Curr;
  int ind;

  if ((NewElement = malloc(sizeof(CONTACT))) == NULL)
    return 0;

  strcpy(NewElement->PhoneNum, NewPhoneNum);
  strcpy(NewElement->SurName, NewSurName);

  ind = NewElement->SurName[0] - 'A';

  if (ind < 0 || ind > 25)
    return 0;

  Curr = &Base[ind].Head;

  OldFirst = NULL;

  while ((*Curr) != NULL)
  {
    OldFirst = *Curr;
    Curr = &(*Curr)->Next;
  }

  NewElement->Next = NULL;
  if (OldFirst != NULL)
    OldFirst->Next = NewElement;
  NewElement->Prev = OldFirst;
  *Curr = NewElement;
  return 1;
}

void DelFromList( char *OldSurName, char *OldPhoneNum )
{
  CONTACT *Prev = NULL, *Curr;
  int ind;

  ind = OldSurName[0] - 'A';

  if (ind < 0 || ind > 25)
    return;

  Curr = Base[ind].Head;

  while (Curr != NULL)
  {
    if (strcmp(Curr->SurName, OldSurName) == 0 && strcmp(Curr->PhoneNum, OldPhoneNum) == 0)
    {
      if (Prev == NULL)
      {
        Base[ind].Head = Curr->Next;
        Base[ind].Head->Prev = Curr->Prev;
```

```
      }
      else
      {
        Prev->Next = Curr->Next;
        if (Curr->Next != NULL)
          Curr->Next->Prev = Curr->Prev;
      }
      free(Curr);
      Curr = NULL;
      return;
    }
    Prev = Curr;
    Curr = Curr->Next;
  }
}

char * FindContact( char *SurName )
{
  int ind;
  CONTACT *Curr;

  ind = SurName[0] - 'A';

  if (ind < 0 || ind > 25)
    return NULL;

  Curr = Base[ind].Head;

  while (Curr != NULL)
  {
    if (strcmp(Curr->SurName, SurName) == 0)
      return Curr->PhoneNum;
    Curr = Curr->Next;
  }
  return NULL;
}

int SaveDB( char *FileName )
{
  FILE *F;
  int i;
  CONTACT *Curr;

  if ((F = fopen(FileName, "wb")) == NULL)
    return 0;

  for (i = 0; i < 26; i++)
  {
    Curr = Base[i].Head;
    while (Curr != NULL)
    {
      fwrite(Curr, sizeof(CONTACT), 1, F);
      Curr = Curr->Next;
    }
  }

  fclose(F);
  return 1;
}

int LoadDB( char *FileName )
{
  FILE *F;
  CONTACT Curr;

  if ((F = fopen(FileName, "rb")) == NULL)
    return 0;
```

```
  while (!feof(F))
  {
    if (fread(&Curr, sizeof(Curr), 1, F) == 1)
      AddToListBack(Curr.SurName, Curr.PhoneNum);
  }

  fclose(F);
  return 1;
}

void SortList( char C )
{
  int ind;
  CONTACT *Curr;

  ind = C - 'A';

  if (ind < 0 || ind > 25)
    return;

  Curr = Base[ind].Head;

  while (Curr != NULL)
  {
    CONTACT *min = Curr, *tmp = Curr->Next;

    while (tmp != NULL)
    {
      if (strcmp(min->SurName, tmp->SurName) > 0)
        min = tmp;
      tmp = tmp->Next;
    }

    if (min != Curr)
    {
      char tmpSN[MAX_NAME_SIZE], tmpPN[PHONE_LENGTH];

      strcpy(tmpSN, Curr->SurName);
      strcpy(tmpPN, Curr->PhoneNum);

      strcpy(Curr->SurName, min->SurName);
      strcpy(Curr->PhoneNum, min->PhoneNum);

      strcpy(min->SurName, tmpSN);
      strcpy(min->PhoneNum, tmpPN);
    }

    Curr = Curr->Next;
  }
}

void ClearList( void )
{
  int i;

  for (i = 0; i < 26; i++)
  {
    CONTACT **Curr = &Base[i].Head;

    if (*Curr == NULL)
      return;
    while ((*Curr)->Prev != NULL)
      Curr = &(*Curr)->Prev;
    while (*Curr != NULL)
    {
      CONTACT *tmp;

      tmp = *Curr;
```

```
      *Curr = (*Curr)->Next;
      if (*Curr == NULL)
      {
        free(tmp);
        return;
      }
      else
        free(tmp);
    }
  }
}
```

main.c:

```
/* Dvas Pavel, 5030102/20002 */

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

#include "list2.h"

void main( void )
{
  InitList();
  LoadDB("ab.db");
  AddToList("Dfff", "+71211804560");
  AddToList("Dvas", "+79211804560");
  AddToList("Danasenko", "+88005553535");
  AddToList("Aarr", "+123123123");
  AddToList("B", "+123123");
  AddToList("FFf", "+12315667");
  //DelFromList("Dvas", "+79211804560");
  SortList('D');
  FindContact("Dvas");
  FindContact("Danasenko");
  SaveDB("ab.db");
  ClearList();
  _getch();
}
```

Описание тестирования

Для тестирования данной программы производился ее неоднократный запуск с вводом конкретных данных, при котором проверялось:

1) Стабильность работы программы при одинаковых входных данных;
2) Отсутствие “падений” и “зависаний”;
3) Корректное выполнение всех заявленных процедур;
4) Корректное завершение программы;

Пример тестирования на скриншотах не приведен, в силу того, что нет материла, который можно было бы скриншотить.

При проведении тестирования такого рода никаких проблем обнаружено не было, что позволяет судить о корректности работы программы в целом.