Лабораторная работа №4. Динамические структуры данных

Часть I. Динамический массив

Задание 1: Разработайте консольное приложение «Телефонная книга», демонстрирующее работу с динамическим массивом структур.

1. Создайте новое решение **Лабораторная работа 4** и проект **Example1** в нем.
2. В проект добавьте заголовочный файл **MyFriendsLib.h**.
3. В заголовочном файле объявите тип структуры и опишите прототипы функций для работы с динамическим массивом:

```
#include <iostream>
#include <string>
#include <fstream>

using namespace std;

//структура «запись в телефонной книге»
struct myFriend
{
   string name;    //имя
   string adress;  //адрес
   int tel;        //телефон
};

//прототипы функций
myFriend *load_book (int &n); //чтение данных из файла и запись в массив
myFriend *new_friend (myFriend * myFriend, int &n);//добавление новой записи в книгу
void print_book (const myFriend *f_book, int n); //вывод на экран всех записей книги
void save_book (const myFriend *f_book, int n);  //сохранение изменений в книге
```

4. В проект добавьте файл исходного кода **MyFriendsLib.cpp**.
5. В файле **MyFriendsLib.cpp** опишите реализацию функций для работы с динамическим массивом:

```
#include "MyFriendsLib.h"

//чтение данных из файла и запись в массив
//n - количество записей в файле, результат - указатель на массив
myFriend * load_book(int &n)
{
   n=0;
   myFriend *f_book;

   ifstream fin ("friends.txt");     //открытие файла для чтения
   if (!fin)                         //если ошибка - выход из программы
   {
         cout << "Невозможно открыть файл!" << endl ;
         exit (-1);
   }
   else
   {
         fin >> n;                     //считывание количества записей в файле
         f_book = new myFriend [n];    //выделение памяти под массив

         //чтение данных из файла в массив
         for(int i=0; i<n; i++)
         {
               fin >> f_book[i].name;
```



```
                fin >> f_book[i].adress;
                fin >> f_book[i].tel;
            }
    }
    fin.close();                        //закрытие файлового потока
    return f_book;                      //возврат указателя на массив
}

//добавление новой записи в книгу
myFriend *new_friend (myFriend * f_book, int &n)
{
    n++; //увеличиваем количество записей

    //Создаем копию массива структур
    struct myFriend * copy_book = new myFriend [n];
    //поэлементно выполняем копирование данных
    for (int i=0; i<n-1; i++)
        copy_book[i]=f_book[i];

    //считываем новые данные
    cout << "Введите имя: ";
    cin >> copy_book[n-1].name;
    cout << endl << "Введите адрес: ";
    cin >> copy_book[n-1].adress;
    cout << endl << "Введите телефон: ";
    cin >> copy_book[n-1].tel;
    delete [] f_book; //освобождаем память "старого" массива
    return copy_book; //возвращаем указатель на "новый" массив
}

//сохранение изменений в книге
void save_book (const myFriend *f_book, int n)
{
    ofstream fout ("friends.txt");  //создание файлового потока для записи
    fout << n << endl;              //запись в файл кол-ва элементов в массиве

    //поэлементный вывод массива в файл
    for (int i=0; i<n; i++)
    {
        fout << f_book[i].name << endl;
        fout << f_book[i].adress << endl;
        fout << f_book[i].tel << endl;
    }
    fout.close();     //закрытие файлового потока
    delete [] f_book; //освобождение памяти
}

//вывод на экран всех записей книги
void print_book (const myFriend *f_book, int n)
{
    //поэлементно выводим на экран информацию из массива
    for (int i = 0; i<n; i++)
    {
        cout << "Запись N" << i+1 << ": " << endl;
        cout << " Имя: " << f_book[i].name << endl;
        cout << " Адрес: " << f_book[i].adress << endl;
        cout << " Телефон: " << f_book[i].tel << endl << endl;
    }
}
```

6. В файле исходного кода **Source.cpp** объявите следующие переменные:

```
myFriend * f_book; //указатель на массив структур - телефонная книга
int n = 0;         //текущее количество записей в книге
```

7. В главной функции **Source.cpp** реализуйте текстовое меню, обеспечивающее выполнение операций с динамическим массивом:

```
f_book=load_book(n); //загрузка данных из файла

int var;             //выбор операции с книгой
do
{
   system("cls");
   cout << "Выберите действие с книгой" << endl;
   cout << "Вывод данных - 1, добавление записи - 2, выход - 0" << endl;
   cin >> var;
   switch (var)
   {
         case 0: cout << "Завершение работы" << endl;
               save_book(f_book, n);
               break;
         case 1: print_book(f_book, n);
               break;
         case 2: f_book=new_friend(f_book, n);
               break;
         default: cout << "Неверный ввод" << endl;
               break;
   }
   system("pause");
} while (var!=0);
```

8. В папке с проектом создайте текстовый файл **friends.txt**, в котором в первой строке будет указано количество записей, а далее будут перечислены данные этих записей, например:

```
3
Masha
Murmansk
133456
Kolja
Novorossijsk
456456
Luda
Vologda
789741
```

9. Запустите программу и проверьте правильность ее работы.

Часть 2. Динамические структуры данных

Задание 2: Разработайте приложение, демонстрирующее работу со стеком.

1. В новом проекте **Example2** создайте заголовочный файл **MyStackLib.h** с предварительными объявлениями:

```
#include <stdlib.h>
#include <iostream>
using namespace std;

//объявление структуры
struct stack
{
   int data;            //данные
   struct stack * next; //ссылка на следующий элемент
};

//функция добавления в стек
struct stack * push (struct stack * head, int n);
//проверка, не пуст ли стек
bool isEmpty (struct stack * head);
//функция снятия элемента со стека
int pop (struct stack * &head);
//вывод на экран всего содержимого стека
void printStack(struct stack * p);
//функция чтения ВЕРХНЕГО элемента стека
int readHead(struct stack * head);
//функция удаления стека
void deleteStack(struct stack * p);
```

2. Далее добавьте в проект файл исходного кода **MyStackLib.cpp** с реализацией функций для работы со стеком:

```
#include "MyStackLib.h"

//функция добавления в стек
//head - указатель на стек, n - добавляемое в стек значение
struct stack * push (struct stack * head, int n)
{
   struct stack * element;     //объявление указателя на новый элемент
   element = new struct stack; //выделение памяти под новый элемент
   element->next=head;         //указание ссылки на предыдущий элемент
   element->data=n;            //задание хранимых данных
   return element;
}

//проверка, не пуст ли стек
//head - указатель на стек
bool isEmpty (struct stack * head)
{
   if (head==NULL)     //если верхний элемент является пустым указателем
         return true;  //стек пуст
   else                //иначе - не пуст
         return false;
}

//функция снятия со стека
int pop (struct stack * &head)
{
   int p; //данные из удаляемого элемента
   if (!isEmpty(head)) //если стек не пуст
   {
```



```
            struct stack * buf; //создаем новый элемент
            buf=head;           //копируем в него текущий верхний элемент
            p=head->data;       //запоминаем данные удаляемой ячейки
            head=head->next;    //удаляем ссылку
            delete(buf);        //освобождаем память
            return p;           //возвращаем данные из удаленной ячейки
      }
      return NULL; //если стек пустой, возвращаем нулевой указатель
   }

   //вывод на экран всего содержимого стека
   void printStack(struct stack * p)
   {
      while (p!=NULL) //пока не конец стека
      {
            cout << p->data << endl; //выводим текущий элемент
            p=p->next;               //переходим по следующему адресу
      }
   }

   //функция чтения ВЕРХНЕГО элемента стека
   int readHead(struct stack * head)
   {
      int p;
      p=head->data;
      return p;
   }

   //функция удаления стека
   void deleteStack(struct stack * p)
   {
      struct stack * buf;
      while (p!=NULL)
      {
            buf=p;       //считываем текущий элемент
            p=p->next;   //переводим ссылку на следующий
            delete(buf); //удаляем текущий
      }
   }
```

3. В файле исходного кода **Source.cpp** выполните работу с элементами стека:

```
   struct stack *head = NULL;  //указатель на верхний элемент
   int n;
   cout << "Введите количество элементов:" << endl;
   cin >> n;

   //занесли в стек числа от 0 до n-1
   for (int i=0; i<n; i++)
      head = push(head,i);

   cout << endl << "Содержимое стека:" << endl;
   printStack(head);           //вывели элементы стека на экран
   cout << "Верхний элемент = " << readHead(head) << endl;

   cout << "Функцией pop сняли верхний элемент." << pop(head) << endl;
   cout << "Теперь верхний элемент = " << readHead(head) << endl;

   cout << "Добавили функций push новый элемент." << endl;
   push (head,10);
   cout << "Содержимое стека изменилось:" << endl;
   printStack(head);
   deleteStack(head);          //удалили стек
```

4. Запустите программу и проверьте правильность ее работы.

Задания для самостоятельной работы

Задание 3: Дополните программу из первого задания функциями поиска записи по имени и удаления записи по ее индексу в массиве.

Задание 4: Реализуйте программу, демонстрирующую работу с односвязным списком.