Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа 9

По дисциплине «Основы алгоритмизации программирования»

На тему «Двусвязнные списки»

Выполнил:

Студент 1 курса 6 группы

Кучерук Николай Петрович

Преподаватель: асс. Андронова М.В.

2023, Минск

**Задание**

| **№ варианта** | **Условие задачи** |
| --- | --- |
| **9** | Разработать функцию, которая формирует стек **Stack**, включив в него по одному разу элементы, которые входят в один из стеков **Stack1** и **Stack2**, но не входят в другой. |

|  |
| --- |
| **Код программы** |
| #include <iostream>  #include <fstream>  using namespace std;  struct node//элемент двусвязного списка  {  int data;//информация, хранящаяся в элементе  node\* next;//указатель на следующий элемент списка  node\* prev;//указатель на прошлый элемент списка  node(int data) {//конструктор элемента  this->data = data;  this->next = NULL;  this->prev = NULL;  }  };  struct list//структура двусвязного списка  {  node\* head, \* tail;//указатели на первый и последний элемент списка  list() {//конструктор списка  this->head = this->tail = NULL;  }  ~list() {//т.к список является динамичной структурой, то для предотвращения утечки памяти создаем деструктор  while (head != NULL) pop\_front();  }  void pop\_front() {//метод для удаления элемента в начале списка  if (head == NULL) return;  if (head == tail) {  delete tail;  head = tail = NULL;  return;  }  node\* node = head;  node->next->prev = NULL;  head = node->next;  delete node;  }  void push\_front(float data) {//метод для добавления элемента в начало списка  node\* Node = new node(data);  if (head == NULL) {  head = tail = Node;  return;  }  head->prev = Node;  Node->next = head;  head = Node;    }  void push\_back(float data) {//метод для добавления элемента в начало списка  node\* Node = new node(data);  if (head == NULL) {  head = tail = Node;  return;  }  tail->next = Node;  Node->prev = tail;  tail = Node;  }  void pop\_id(int number) {//метод для удаления элементов номера на выбор пользователя  node\* Node = head;  int counter = 0;  while (Node != NULL) {  if (counter == number)  {  if (head == Node) { head = Node->next; }  if (Node->next == NULL) { tail = Node->prev; }  Node->prev->next = Node->next;  Node->next->prev = Node->prev;  delete Node;  return;  }  Node = Node->next;  counter++;  }  }  void AddBackX(int x) {//метод для переноса элементов в конец  node\* Node = head;  int counter = 0;  while (Node != NULL) {  if (counter == x-1)  {  this->push\_back(Node->data);  this->pop\_id(counter);  return;  }  Node = Node->next;  counter++;  }  }  void output() {//метод для вывода списка  for (node\* Node = head; Node != NULL; Node = Node->next) {  cout << Node->data << " ";  }  cout << endl;  }  };  void menu(void); //функция, показывающая меню  void filewr(list&); //функция записи списка в файл  void filerd(list&); //функция получения списка из файла  int main()  {  setlocale(LC\_CTYPE, "Russian");  list first;//создаем список  int choice;  float value;  menu(); // вывести меню  cout << " ? ";  cin >> choice;  while (choice != 5)  {  switch (choice)  {  case 1: cout << "Введите число " << endl; // добавить число в список  cin >> value;  first.push\_front(value);  cout << "Cписок: ";  first.output();  break;  case 2:  {  cout << "Введите номер переносимого элемента ";  cin >> value;  first.AddBackX(value);  cout << "Новый список ";  first.output();  }  break;  case 3:  {  filewr(first);  cout << "Файл записан " << endl;  break;  }  case 4:  {  filerd(first);  cout << "Данные из файла переписаны в список " << endl;  first.output();  break;  }  default: cout << "Неправильный выбор" << endl;  menu();  break;  }  cout << "? ";  cin >> choice;  }  cout << "Конец" << endl;  return 0;  }  void menu(void) //Вывод меню  {  cout << "Сделайте выбор:" << endl;  cout << " 1 - Ввод числа" << endl;  cout << " 2 - Перенос элемента в конец" << endl;  cout << " 3 - Ввод в файл" << endl;  cout << " 4 - Вывод из файла" << endl;  cout << " 5 - Выход" << endl;  }  void filewr(list& p) {//запись данных списка в файл  ofstream file("list.txt");  for (node\* Node = p.head; Node != NULL; Node = Node->next) {  file << Node->data << " ";  }  file.close();  }  void filerd(list& temp) {//чтение данных из файла в список  ifstream file("list.txt");  if (file.fail())  {  cout << "\n File cannot be opened";  exit(1);  }  while (temp.head != NULL) temp.pop\_front();  float a;  while (file >> a) {  temp.push\_front(a);  }  file.close();  } |
| **Результат программы** |
|  |

Дополнительные задания

1. В [приложении 2](#prilogenie2) приведен проект, в котором реализован проект с использованием *двусвязного* *списка*. На основе данного проекта разработать функции, которые предлагается создать в данном приложении.

|  |
| --- |
| **Код программы** |
| **OAP\_list.cpp**  #include "list\_header.h"  void print(void\* b) {  Country\* a = (Country\*)b;  cout << a->name << " " << a->capital << " " << a->population << " " << a->area << " " << a->head\_lastname << endl;  }  int main() {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  system("color 70");  Object L1 = Create();  unsigned short choice;  bool rc;  do {  cout << "\n0. Выход из приложения\n";  cout << "1. Ввод элементов списка с клавиатуры\n";  cout << "2. Вывод элементов списка в консольное окно\n";  cout << "3. Удаление заданного элемента\n";  cout << "4. Поиск информации\n";  cout << "5. Расчет элементов списка\n";  cout << "6. Удаление списка\n";  cout << "Введите вариант работы приложения: ";  cin >> choice;  switch (choice)  {  case 1: { //ввод элементов структуры  Country\* newCountry = new Country;  cout << "Название страны: ";  cin >> newCountry->name;  cout << "Столица страны: ";  cin >> newCountry->capital;  cout << "Численность населения страны: ";  cin >> newCountry->population;  cout << "Площадь страны: ";  cin >> newCountry->area;  cout << "Фамилия главы страны: ";  cin >> newCountry->head\_lastname;  L1.Insert(newCountry);  break;  }  case 2: { //вывод элементов структуры  cout << "Список:" << endl;  L1.PrintList(print); // выводим новый список  break;  }  case 3: { //удаление заданного элемента  char deleteName[20];  cout << "Введите необходимую страну: ";  cin >> deleteName;  rc = L1.Delete(deleteName);  if (rc) {  cout << "Элемент удален" << endl;  }  else {  cout << "Элемент не найден" << endl;  }  break;  }  case 4: { //поиск  char searchName[20];  cout << "Введите необходимую страну: ";  cin >> searchName;  Element\* find = L1.Search(searchName);  if (find != NULL) {  Country\* countryInfo = (Country\*)find->Data;  cout << "Информация о стране: " << endl;  print(countryInfo);  }  else {  cout << "Страна не найдена" << endl;  }  break;  }  case 5: { // подсчет элементов списка  cout << "Количество элементов в списке: " << L1.countList() << endl;  break;  }  case 6: { // удаление списка  rc = L1.deleteList(); // Удаляем элементы списка  if (rc) {  cout << "Список удален" << endl;  }  else {  cout << "Список пуст!" << endl;  }  break;  }  case 0: { //выход из программы  exit(0);  break;  }  default: {  cout << "Введено некорректное значение. " << endl;  break;  }  }  } while (choice != 0);  return 0;  }  **List.cpp**  #include "list\_header.h"  Object::~Object() {  while (Head != NULL) { // пока есть элементы в списке  Element\* buffer = Head;  Head = Head->Next;  delete buffer;  }  }  Element\* Object::GetLast() {  Element\* buffer = Head;  Element\* result = buffer;  while (buffer != NULL) { // ищем до последнего элемента  result = buffer;  buffer = buffer->Next;  }  return result;  }  Element\* Object::Search(char\* data) {  Element\* result = Head;  while ((result != NULL) && (strcmp((char\*)result->Data, data) != 0)) { // ищем до нужного элемента  result = result->Next;  }  return result;  }  void Object::Insert(Country\* data) {  if (Head == NULL) { // просто создаем новый элемент,если список пуст  Head = new Element(NULL, data, NULL);  }  else { // добавляем новый элемент в конец списка,если в списке уже есть элементы  Element\* last = GetLast(); // Получаем последний элемент  last->Next = new Element(last, data, NULL);  }  }  bool Object::Delete(Element\* buffer) {  if (buffer == NULL) return false; // если список пуст  if (buffer->Next != NULL)  buffer->Next->Prev = buffer->Prev; // обновляем указатели соседних элементов  if (buffer->Prev != NULL)  buffer->Prev->Next = buffer->Next;  else  Head = buffer->Next;//при удалении текущего head  delete buffer;  return true;  }  bool Object::Delete(char\* data) {  return Delete(Search(data));  }  void Object::PrintList(void(\*f)(void\*)) {  Element\* buffer = Head;  while (buffer != NULL) { // Пока есть элементы в списке  f(buffer->Data); // Вызываем функцию для вывода данных элемента  buffer = buffer->GetNext(); // Переходим к следующему элементу  }  }  void Object::PrintList(void(\*f)(void\*), Element\* buffer) {  f(buffer->Data); // Вызываем функцию для вывода данных элемента  }  int Object::countList() {//счетчик элементов в списке  int count = 0;  Element\* buffer = Head;  while (buffer != NULL) { // Пока есть элементы в списке  count++; // Вызываем функцию для вывода данных элемента  buffer = buffer->GetNext();// Переходим к следующему элементу  }  return count;  }  bool Object::deleteList() { // Удаляем список  if (Head == NULL) {  return false;  }  while (Head != NULL) {  Element\* buffer = Head;  Head = Head->GetNext();  delete buffer;  }  return true;  }  // конструтор списка  Object Create() {  return Object();  }  **List.h**  #pragma once  #include <iostream>  #include <Windows.h>  using namespace std;  struct Country {  char name[20];  char capital[20];  unsigned int population;  double area;  char head\_lastname[20];  Country\* next;  };  struct Element {//единица списка  Element\* Prev;  Element\* Next;  void\* Data;  Element(Element\* prev, void\* data, Element\* next) {  Prev = prev;  Data = data;  Next = next;  }  Element\* GetNext() {  return Next;  };  Element\* GetPrev() {  return Prev;  };  };  struct Object {//структура списка  Element\* Head;  Object() {  Head = NULL;  };  ~Object(); // деструктор  Element\* GetFirst() {  return Head;  };  Element\* GetLast();  Element\* Search(char\* data);  void Insert(Country\* data);  bool Delete(char\* data);  bool Delete(Element\* data);  void PrintList(void(\*f)(void\*));  void PrintList(void(\*f)(void\*), Element\*);  int countList();// подсчет числа элементов списка  bool deleteList();//удаление списка  };  Object Create(); |
| **Результат программы** |
|  |

2. Каждый элемент списка студентов содержит фамилию, имя, отчество, год рождения, курс, номер группы, оценки по пяти предметам. Упорядочить студентов по курсу, причем студенты одного курса должны располагаться в алфавитном порядке. Найти средний балл каждой группы по каждому предмету. Определить самого старшего студента и самого младшего. Для каждой группы найти лучшего с точки зрения успеваемости студента.

|  |
| --- |
| **Код программы** |
| #include <iostream>  #include <Windows.h>  #include <iomanip>  #include <string>  using namespace std;  struct node//элемент двусвязного списка  {  string last\_name;  string name;  string surname;  int birth\_year;  int course;  int group;  int grades[5];    node\* next; //следующий человек в списке  node\* prev; //предыдущий человек в списке  node(string lname, string name, string surname,int bd,int course, int group,int grades[5]) {//конструктор элемента  this->last\_name = lname;  this->name = name;  this->birth\_year = bd;  this->course = course;  this->group = group;  for (int i = 0; i < 5;i++) {  this->grades[i] = grades[i];  }  this->next = NULL;  this->prev = NULL;  }  };  struct grp\_stat {  short group\_num; //Номер группы  short total[5]; //Сумма баллов по каждому предмету  short amount; // Количество студентов в группе  };  struct list//структура двусвязного списка  {  node\* head, \* tail;//указатели на первый и последний элемент списка  list() {//конструктор списка  this->head = this->tail = NULL;  }  ~list() {//т.к список является динамичной структурой, то для предотвращения утечки памяти создаем деструктор  while (head != NULL) pop\_front();  }  void pop\_front() {//метод для удаления элемента в начале списка  if (head == NULL) return;  if (head == tail) {  delete tail;  head = tail = NULL;  return;  }  node\* node = head;  node->next->prev = NULL;  head = node->next;  delete node;  }  void push\_front(string lname, string name, string surname, int bd, int course, int group, int grades[5]) {//метод для добавления элемента в начало списка  node\* Node = new node( lname, name, surname, bd, course, group, grades);  if (head == NULL) {  head = tail = Node;  return;  }  head->prev = Node;  Node->next = head;  head = Node;    }  void pop\_id(int number) {//метод для удаления элементов содержащих число на выбор пользователя  node\* Node = head;  int counter = 0;  while (Node != NULL) {  if (counter == number)  {  if (head == Node) { head = Node->next; }  if (Node->next == NULL) { tail = Node->prev; }  Node->prev->next = Node->next;  Node->next->prev = Node->prev;  delete Node;  return;  }  Node = Node->next;  counter++;  }  }    void output() {//метод для вывода списка  for (node\* Node = head; Node != NULL; Node = Node->next) {  cout << Node->last\_name << " " << Node->name << " " << Node->surname << " " << Node->birth\_year << " " << Node->course << " "<<Node->group<<' ';  for (int i = 0; i < 5; i++) {  cout << Node->grades[i] << " ";  }  cout << endl;  }  cout << endl;  }  void sort() {  if (head->next == nullptr) {  // Список содержит только один элемент, сортировка не требуется  return;  }  node\* current = head->next; // Начинаем со второго элемента  while (current != nullptr) {  node\* temp = current;  while (temp->prev != nullptr && (temp->course < temp->prev->course ||(temp->course == temp->prev->course && temp->last\_name.compare( temp->prev->last\_name) < 0))) {  // Сравниваем курсы и фамилии и меняем местами студентов в списке  node\* prevNode = temp->prev;  node\* nextNode = temp->next;  temp->prev = prevNode->prev;  temp->next = prevNode;  prevNode->prev = temp;  prevNode->next = nextNode;  if (nextNode != nullptr) {  nextNode->prev = prevNode;  }  if (temp->prev != nullptr) {  temp->prev->next = temp;  }  else {  // Обновляем указатель на начало списка  head = temp;  }  }  current = current->next;  }  tail = head;  }  //Нахождение среднего балла в каждой группе по каждому предмету  void average( unsigned int n) {  grp\_stat\* groups = new grp\_stat[n];  for (int i = 0; i < n; i++) {  groups[i].group\_num = i + 1;  groups[i].amount = 0;  for (int j = 0; j < 5; j++) {  groups[i].total[j] = 0;  }  }  node\* temp = head;  while (temp != nullptr) {  int groupIndex = temp->group - 1; // Индекс группы в массиве  groups[groupIndex].amount++;  for (int i = 0; i < 5; i++) {  groups[groupIndex].total[i] += temp->grades[i];  }  temp = temp->next;  }  cout << "Средние баллы группы по каждому предмету: " << endl;  for (int i = 0; i < n; i++) {  cout << "Группа " << groups[i].group\_num << ": ";  for (int j = 0; j < 5; j++) {  double average = groups[i].amount > 0 ? static\_cast<double>(groups[i].total[j]) / groups[i].amount : 0.0;  cout << fixed << setprecision(2) << average << " ";  }  cout << endl;  }  delete[] groups;  }  void young\_and\_old() {  node\* temp = head;  node\* youngest = temp;  node\* oldest = temp;  while (temp != nullptr) {  if (temp->birth\_year > youngest->birth\_year) {  youngest = temp;  }  if (temp->birth\_year < oldest->birth\_year) {  oldest = temp;  }  temp = temp->next;  }  cout << "Самый младший студент: " << youngest->last\_name << " " << youngest->name << " " << youngest->surname << " (" << youngest->birth\_year << ")" << std::endl;  cout << "Самый старший студент: " << oldest->last\_name << " " << oldest->name << " " << oldest->surname << " (" << oldest->birth\_year << ")" << std::endl;  }  void best\_performance() {  node\* bestStudent = head;  node\* temp = head;  while (temp != nullptr) {  int tempSum = 0;  for (int i = 0; i < 5; i++) {  tempSum += temp->grades[i];  }  int bestSum = 0;  for (int i = 0; i < 5; i++) {  bestSum += bestStudent->grades[i];  }  if (tempSum > bestSum) {  bestStudent = temp;  }  temp = temp->next;  }  cout << "Лучший студент:" << endl;  cout << bestStudent->last\_name << " " << bestStudent->name << " " << bestStudent->surname << endl;  cout << "Год рождения: " << bestStudent->birth\_year << endl;  cout << "Курс: " << bestStudent->course << endl;  cout << "Группа: " << bestStudent->group << endl;  cout << "Оценки: ";  for (int i = 0; i < 5; i++) {  cout << bestStudent->grades[i] << " ";  }  cout << endl;  }    };  void input(list&); //функция, показывающая меню  int main()  {  system("color 70");  setlocale(LC\_CTYPE, "Russian");  list first;//создаем список  int n;  cout << "Сколько хотите ввести студентов? - ";  cin >> n;  for (int i = 0; i < n; i++) {  input(first);  }  cout << "Полученный список: " << endl;  first.sort();  first.output();  first.average(n);  first.young\_and\_old();  first.best\_performance();  }  void input(list& first) {  string lname, name, sname;  int bd, grades[5], group, course;  cout << "Введите фамилию: ";  getline(cin, lname);  getline(cin, lname);  cout << "Введите имя: ";  cin.clear();  getline(cin, name);  cout << "Введите отчество: ";  cin.clear();  getline(cin, sname);  cout << "Введите год рождения: ";  cin >> bd;  cout << "Введите курс: ";  cin >> course;  cout << "Введите группу: ";  cin >> group;  cout << "Введите оценки (5): ";  for (int i = 0; i < 5; i++) {  cin >> grades[i];  }  first.push\_front(lname, name, sname, bd, course, group, grades);  cout << "Cписок: ";  first.output();  } |
| **Результат программы** |
|  |

|  |  |
| --- | --- |
| **10** | **addXBegin (int x)** – функция добавления элемента **x** в начало списка. |

|  |
| --- |
| **Код программы** |
| #include <iostream>  #include <fstream>  using namespace std;  struct node//элемент двусвязного списка  {  int data;//информация, хранящаяся в элементе  node\* next;//указатель на следующий элемент списка  node\* prev;//указатель на прошлый элемент списка  node(int data) {//конструктор элемента  this->data = data;  this->next = NULL;  this->prev = NULL;  }  };  struct list//структура двусвязного списка  {  node\* head, \* tail;//указатели на первый и последний элемент списка  list() {//конструктор списка  this->head = this->tail = NULL;  }  ~list() {//т.к список является динамичной структурой, то для предотвращения утечки памяти создаем деструктор  while (head != NULL) pop\_front();  }  void pop\_front() {//метод для удаления элемента в начале списка  if (head == NULL) return;  if (head == tail) {  delete tail;  head = tail = NULL;  return;  }  node\* node = head;  node->next->prev = NULL;  head = node->next;  delete node;  }  void push\_front(float data) {//метод для добавления элемента в начало списка  node\* Node = new node(data);  if (head == NULL) {  head = tail = Node;  return;  }  head->prev = Node;  Node->next = head;  head = Node;    }  void push\_back(float data) {//метод для добавления элемента в начало списка  node\* Node = new node(data);  if (head == NULL) {  head = tail = Node;  return;  }  tail->next = Node;  Node->prev = tail;  tail = Node;  }  void pop\_id(int number) {//метод для удаления элементов номера на выбор пользователя  node\* Node = head;  int counter = 0;  while (Node != NULL) {  if (counter == number)  {  if (head == Node) { head = Node->next; }  if (Node->next == NULL) { tail = Node->prev; }  Node->prev->next = Node->next;  Node->next->prev = Node->prev;  delete Node;  return;  }  Node = Node->next;  counter++;  }  }  void AddFrontX(int x) {//метод для переноса элементов в конец  node\* Node = head;  int counter = 0;  while (Node != NULL) {  if (counter == x-1)  {  this->push\_front(Node->data);  this->pop\_id(counter);  return;  }  Node = Node->next;  counter++;  }  }  void output() {//метод для вывода списка  for (node\* Node = head; Node != NULL; Node = Node->next) {  cout << Node->data << " ";  }  cout << endl;  }  };  void menu(void); //функция, показывающая меню  void filewr(list&); //функция записи списка в файл  void filerd(list&); //функция получения списка из файла  int main()  {  setlocale(LC\_CTYPE, "Russian");  list first;//создаем список  int choice;  float value;  menu(); // вывести меню  cout << " ? ";  cin >> choice;  while (choice != 5)  {  switch (choice)  {  case 1: cout << "Введите число " << endl; // добавить число в список  cin >> value;  first.push\_front(value);  cout << "Cписок: ";  first.output();  break;  case 2:  {  cout << "Введите номер переносимого элемента ";  cin >> value;  first.AddFrontX(value);  cout << "Новый список ";  first.output();  }  break;  case 3:  {  filewr(first);  cout << "Файл записан " << endl;  break;  }  case 4:  {  filerd(first);  cout << "Данные из файла переписаны в список " << endl;  first.output();  break;  }  default: cout << "Неправильный выбор" << endl;  menu();  break;  }  cout << "? ";  cin >> choice;  }  cout << "Конец" << endl;  return 0;  }  void menu(void) //Вывод меню  {  cout << "Сделайте выбор:" << endl;  cout << " 1 - Ввод числа" << endl;  cout << " 2 - Перенос элемента в начало" << endl;  cout << " 3 - Ввод в файл" << endl;  cout << " 4 - Вывод из файла" << endl;  cout << " 5 - Выход" << endl;  }  void filewr(list& p) {//запись данных списка в файл  ofstream file("list.txt");  for (node\* Node = p.head; Node != NULL; Node = Node->next) {  file << Node->data << " ";  }  file.close();  }  void filerd(list& temp) {//чтение данных из файла в список  ifstream file("list.txt");  if (file.fail())  {  cout << "\n File cannot be opened";  exit(1);  }  while (temp.head != NULL) temp.pop\_front();  float a;  while (file >> a) {  temp.push\_front(a);  }  file.close();  } |
| **Результат программы** |
|  |