|  |
| --- |
| Студент: Пантюхин А.Е., группа: ДТ-460а |
| Лабораторная работа №4 |
| Задание:  Вариант 9  В бинарном файле хранится последовательность целых чисел. Вывести в порядке убывания те числа, которые встречаются в последовательности более одного раза.  Использовать для решения задачи односвязный список.  В рамках решения реализовать функции:   1. Ввода чисел с клавиатуры и записи введённого в файл 2. Чтения неупорядоченных чисел из файла и вывода в поток stdout 3. Создания списка из упорядоченных по возрастанию/убыванию значений, каждый элемент содержит счётчик повторений числа и само число. 4. Вывод содержимого списка на экран. |
| Тестовые данные:   1. 1 2 2 2 3 -4 -4 5 66 8 8 1 2. 0 3. 3 3 3 3 3 3 3 3 3 3 3 3 3 |
| Текст программы с комментариями:  List\_fill.c:  #include <stdio.h>  #include <stdlib.h>  #include <string.h>    //создание и заполнение бинарного файла числами  int main(void)  {  char user\_input[BUFSIZ] = {0}; //буфер для хранения и обработки пользовательского ввода  printf("Укажите имя файла для вывода списка:\n");  fgets(user\_input, BUFSIZ-1, stdin); //получили всю строку потока stdin  sscanf(user\_input, "%s", user\_input); //получили из строки только значимую часть (до первого пустого элемента (" ", "\n", "\0"))  printf("Вы ввели: %s\n", user\_input); //вывод для проверки  FILE\* file = fopen(user\_input, "wb"); //открытие бинарного файла для записи (и/или его создание)  if (!file)  {  printf("Ошибка! Файл невозможно открыть для записи!\n");  return 1;  }    printf("Введите элементы списка через пробел, ввод новой строки завершит ввод:\n");  fgets(user\_input, BUFSIZ-1, stdin); //вновь получаем строку из stdin  printf("В файл записано: "); //информируем пользователя параллельно с записью  char\* token = strtok(user\_input, " ,"); //"токенизация" строки - отделяем первый элемент (до пробела или запятой)  while (token)  { //пока элемент найден  int value = atoi(token); //получаем значение типа int из этого элемента  printf("%d ", value); //информируем пользователя  fwrite(&value, sizeof(int), 1, file); //запись элемента в бинарный файл  token = strtok(NULL, " ,"); //отделяем от исходной строки следующий элемент  }  printf("\n");  fclose(file);//закрываем поток.  return 0;  }  Task.c:  //В бинарном файле хранится последовательность целых чисел.  // Вывести в порядке убывания те числа, которые встречаются в последовательности  // более одного раза.  // Использовать для решения задачи односвязный список.    #include <stdio.h>    #include "list.h"    /\*  Получаем из указанного бинарного файла данные в виде набора чисел, формируем из них список1  Полученный список1 конвертируем в список1 с учётом количества вхождений  Выводим значения списка2, если их количество вхождений больше 1  \*/  int main(void) {  list\* l1 = 0;//"входной" список1  stacked\_list\* slist1 = 0; //итоговый список2  char user\_input[BUFSIZ] = {};//буфер для пользовательского ввода  //BUFSIZ - размер буфера потока из stdio  printf("Введите имя файла для ввода списка:\n");  scanf("%s", user\_input); //scanf потому, что fgets заберёт \n тоже, и, в отличие от заполнения, нам нет дела до того, что останется в потоке  if (!list\_get\_from\_file(user\_input, &l1)) printf("Ошибка! Невозможно прочитать файл!\n");  else  {  printf("Содержимое файла:\n");  list\_print(l1); //вывод списка, описано в месте определения  printf("\n");  /\*  поздно увидел, что нужна структура с количеством повторений  поэтому конвертируем+сортируем  \*/  list\_to\_slist\_s(l1, &slist1, 0); //конвертация списка, описано в месте определения  printf("Список из значений, встречающихся более одного раза, в порядке убывания: \n");  //вывод всех с кол-вом вхождений больше 1.  slist\_print\_many(slist1); //описано в месте определения  printf("\n");  }  //очистка памяти  list\_dealloc(&l1); //описано в месте определения  slist\_dealloc(&slist1); //описано в месте определения  return 0;  }    List.h:  #ifndef LIST\_H  #define LIST\_H //защита от переопределения    //список  typedef struct list  {  int value; //значение  struct list\* next; //служебная часть  }  list;    //список с подсчётом количества вхождений  typedef struct stacked\_list  {  int value; //значение  int amount; //количество вхождений  struct stacked\_list\* next; //служебная часть  }  stacked\_list;    /\*  функции описаны в файле с определениями.  \*/    void list\_push\_back(int value, list\*\* head);    void list\_insert(int value, list\*\* head);    void list\_add\_sorted(int value, list\*\* head);    void list\_dealloc(list\*\* head);    void list\_print(const list\* head);    const list\* list\_contains(int value, const list\* head);    int list\_contains\_amount(int value, const list\* head);    int list\_get\_from\_file(const char\* path, list\*\* head);    int list\_fill\_file(const char\* path, const list\* head);    stacked\_list\* slist\_get(int value, stacked\_list\* head);    int slist\_insert(int value, stacked\_list\*\* head);    int slist\_add\_s(int value, stacked\_list\*\* head, int is\_raising);    int list\_to\_slist\_s(const list\* src, stacked\_list\*\* dest, int is\_raising);    void slist\_print(const stacked\_list\* head);    void slist\_print\_many(const stacked\_list\* head);    void slist\_dealloc(stacked\_list\*\* head);    #endif  List.c:  #include <stdlib.h>  #include <stdio.h>    #include "list.h"    //добавление элемента в конец списка  void list\_push\_back(int value, list\*\* head)  {  if (!head) return; //некорректные данные, выход  list\* ptr = \*head; //служебные указатели  list\* prev = 0;  while (ptr)  { //движение в конец списка  prev = ptr;  ptr = ptr->next;  }  ptr = malloc(sizeof(list)); //выделение памяти  \*ptr = (list){value, 0}; //инициализация  if (prev) prev->next = ptr; //если есть прошлый элемент  else \*head = ptr; //иначе - мы в начале списка  return;  }    //добавление элемента в начало списка  void list\_insert(int value, list\*\* head)  {  if (!head) return; //некорректные данные, выход  list\* temp = malloc(sizeof(list)); //выделение памяти  \*temp = (list) {value, \*head};//инициализация  \*head = temp; //добавление в список  }    //освобождение памяти под список  void list\_dealloc(list\*\* head)  {  if (!head) return; //некорректные данные, выход  if (!\*head) return; //конец списка(или его просто нет)  list\_dealloc(&((\*head)->next)); //рекурсивный вызов для следующего элемента  free(\*head); //очистка памяти  \*head = 0; //зануление, на случай если \*head будет переиспользован  return;  }    //вывод списка в поток stdout  void list\_print(const list\* head)  {  if (!head) return; //некорректные данные, выход  printf("%d ", head->value); //вывод значения  list\_print(head->next); //рекурсивный вызов для следующего элемента  return;  }    //чтение списка из бинарного файла  int list\_get\_from\_file(const char\* path, list\*\* head)  {  int buffer; //новый элемент списка  FILE\* file = fopen(path, "rb"); //открываем бинарный файл на чтение  if (!file) return 0; //файл не открылся, ошибка  while(fread(&buffer, sizeof(int), 1, file))  { //fread вернёт число считанных элементов, если 0 - значит, файл закончился или встречен некорректный формат(некратный 4)  //пока что-то прочитано, записываем значение в конец списка  list\_push\_back(buffer, head);  }  fclose(file); //закрываем поток.  return 1;//1 == ошибок нет  }    //заполнение бинарного файла набором значений из списка.  int list\_fill\_file(const char\* path, const list\* head)  {  FILE\* file = fopen(path, "wb"); //открываем бинарный файл на чтение(содержимое будет стёрто)  if (!file) return 0; //файл не открылся, ошибка  while (head)  {  //запись значения в файл  fwrite(&(head->value),sizeof(int), 1,file);  //перемещение по списку  head = head->next;  }  fclose(file); //закрываем поток  return 1; //1 == ошибок нет  }    //получить адрес элемента с данным значением в списке  stacked\_list\* slist\_get(int value, stacked\_list\* head)  {  if (!head) return 0; //некорректные данные, выход  if (head->value == value) return head; //если значение равно искомому - возвращаем адрес  return slist\_get(value, head->next); //рекурсивный вызов функции для следующего элемента (возвращаем результат)  }    //добавление элемента в начало списка (или, если он уже есть - увеличить кол-во на 1)  int slist\_insert(int value, stacked\_list\*\* head)  {  if (!head) return 0;//некорректные данные, выход  stacked\_list\* elem = slist\_get(value, \*head);//если существует, получит адрес  if (elem) elem->amount++; //если есть адрес, количество++  else  { //если нет такого элемента  elem = malloc(sizeof(stacked\_list)); //выделяем память  \*elem = (stacked\_list){value, 1, \*head}; //инициализация  \*head = elem; //добавление в начало списка  }  return 1; //1 == ошибок нет  }    int slist\_add\_s(int value, stacked\_list\*\* head, int is\_raising)  {  if (!head) return 0; //некорректные данные, выход  stacked\_list\* ptr = \*head, \*prev = 0; //объявляем и инициализируем служебные переменные  //если список пустой или такой элемент есть, сразу добавляем  if (!ptr || slist\_get(value, ptr)) return slist\_insert(value, head);  //иначе, начинаем искать место для элемента  while (ptr //пока не дошли до конца  //и (по возрастанию) значение элемента меньше входного  && (((ptr->value < value) && is\_raising)  //или (по убыванию) значение элемента больше входного  || ((ptr->value > value) && !is\_raising)))  {  prev = ptr; //запоминаем прошлый элемент  ptr = ptr->next; //движемся по списку  }  //Либо нашли место, и ставим элемент на него, либо мы находимся в начале списка.  slist\_insert(value, &ptr);  //Если не в начале - добавляем элемент в список "справа"  if (prev) prev->next = ptr;  //если в начале - добавляем "слева".  else \*head = ptr;  return 1; //1 == ошибок нет  }    //конвертация списка в список с количеством вхождений  int list\_to\_slist\_s(const list\* src, stacked\_list\*\* dest, int is\_raising)  {  int all\_good = 1; //флаговая переменная, 1 == ошибок нет  while(src && all\_good)  { //пока не в конце исходного списка и нет ошибок...  all\_good = slist\_add\_s(src->value, dest, is\_raising); //добавляем в список с количеством элемент  src = src->next; //движемся по исходному списку  }  return all\_good;  }    //вывод списка с количеством вхождений в формате <значение>[<количество>]...  void slist\_print(const stacked\_list\* head)  {  if (!head) return; //списка нет или мы уже в конце, выход  printf("%d[%d] ", head->value, head->amount); //вывод значения  slist\_print(head->next); //рекурсивный вызов для следующего элемента  return;  }    //вывод элементов, где количество вхождений больше 1.  void slist\_print\_many(const stacked\_list\* head)  {  if (!head) return; //списка нет или мы уже в конце, выход  if (head->amount > 1) printf("%d ", head->value); //вывод значения  slist\_print\_many(head->next); //рекурсивный вызов для следующего элемента  return;  }    //освобождение памяти  void slist\_dealloc(stacked\_list\*\* head)  {  if (!head) return; //некорректные данные, выход  if (!\*head) return; //конец списка, выход  slist\_dealloc(&((\*head)->next)); //рекурсивный вызов для следующего элемента  free(\*head);//освобождение памяти  \*head = 0; //зануление для возможности переиспользования указателя  return;  }    Makefile (использовался для сборки и тестирования проекта):  all: filler task    clean:  @ rm -f liblist.a  @ rm -f list.o  @ rm -f task  @ rm -f filler    liblist.a: list.c list.h  @ gcc list.c -c -o list.o  @ ar rcs liblist.a list.o    filler: list\_fill.c  @ gcc list\_fill.c -o filler    task: task.c liblist.a  @ gcc task.c liblist.a -o task    fill: filler  @ ./filler    run: task  @ ./task |