МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения вычислительной техники

и автоматизированных систем

**КУРСОВАЯ РАБОТА**

по дисциплине:”Основы алгоритмизации и программирования”

на тему: ***”*** ***Алгоритмизация с использованием динамических структур «Списки и очереди». ”***

Выполнил**:** ст. гр. 10701118 Воробей И.А

Приняла**:** ст**.** преподаватель Борисова И.М.

Минск 2019

Белорусский национальный технический университет

Кафедра программного обеспечения вычислительной техники

и автоматизированных систем

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту (работе)**

**по дисциплине\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Тема \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Исполнитель**:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(фамилия, инициалы)

(подпись)

**Студент\_\_\_\_\_\_\_\_\_курса\_\_\_\_\_\_\_\_\_\_группы**

**Руководитель** : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(фамилия, инициалы)

(подпись)

Минск 2019

**Оглавление**

[ВВЕДЕНИЕ 5](#_Toc7504369)

[1 СТЕКИ 6](#_Toc7504370)

[1.1 Теоретический вопрос 6](#_Toc7504371)

[1.2 Постановка задачи 6](#_Toc7504372)

[1.3 Описание программы и её решение 6](#_Toc7504373)

[1.3.1 Блок-схемы алгоритмов 7](#_Toc7504374)

[1.3.2 Словесное описание работы программы 8](#_Toc7504375)

[1.4Тестирование 10](#_Toc7504376)

[2 ЛИНЕЙНЫЕ СПИСКИ 12](#_Toc7504377)

[2.1 Теоретический вопрос 12](#_Toc7504378)

[2.2 Постановка задания 13](#_Toc7504379)

[2.3 Описание программы и её решения 13](#_Toc7504380)

[2.3.1Лок-схемы алгоритмов 14](#_Toc7504381)

[2.3.2 Словесное описание работы программы 17](#_Toc7504382)

[2.4 Тестирование 18](#_Toc7504383)

[ЗАКЛЮЧЕНИЕ 23](#_Toc7504384)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ И ИСТОЧНИКОВ 24](#_Toc7504385)

[ПРИЛОЖЕНИЕ А 25](#_Toc7504389)

[ПРИЛОЖЕНИЕ Б 27](#_Toc7504390)

# ВВЕДЕНИЕ

В связи с переходом к информационному обществу и развитием информационных технологий наша жизнь становится все более мобильной и информативной, новые технологии прочно входят в различные отрасли хозяйствования и сферы жизни. На сегодняшний день компьютеры гораздо быстрее и производительнее, чем были раньше. Вместе с этим растет неудовлетворенность человека в качестве обработки и передаче информации, соответственно увеличивается потребность в более быстрой и эффективной обработке информации. Поэтому для решения различных задач совершенствуются способы хранения и обработки информации.

Тема данной курсовой работы : «Алгоритмизация с использованием динамических структур “Cписки и очереди”».

Довольно часто в процессе работы с различными данными невозможно определить, сколько памяти потребуется для их хранения, поэтому память лучше всего распределять во время выполнения программы отдельными блоками. Блоки связываются друг с другом с помощью указателей. Такой способ организации данных называется динамической структурой данных. Это структуры данных, память под которые выделяется и освобождается по мере необходимости.

Динамическая структура данных характеризуется тем что она не имеет имени, ей выделяется память в процессе выполнения программы, количество элементов структуры может не фиксироваться, размерность структуры может меняться в процессе выполнения программы, в процессе выполнения программы может меняться характер взаимосвязи между элементами структуры.

Таким образом, целью курсовой работы является разработка программы на языке C++, осуществляющей эффективное хранение и обработку данных на основе линейных списков, закрепление и углубление знаний, полученных в курсе " Основы алгоритмизации и программирования ", развитие навыков при выборе представления исходных данных, при написании программ на языке C/С++, тестировании и отладки программы, оформлении документации на программную разработку.

Однако использование динамических структур не лишено имеет и недостатки, основными из которых является то, что на поля, содержащие указатели для связывания элементов друг с другом, расходуется дополнительная память; доступ к элементам связной структуры может быть менее эффективным по времени.

**1 СТЕКИ**

**1.1 Теоретический вопрос**

Стек (англ. stack — стопка; читается стэк) — абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»).[2]Стек состоит из ячеек, которые представлены в виде структуры, содержащей какие-либо данные и указатель типа данной структуры на следующий элемент. Чаще всего принцип работы стека сравнивают со стопкой тарелок: чтобы взять вторую сверху, нужно снять верхнюю.[1]

В [цифровом вычислительном комплексе](https://ru.wikipedia.org/wiki/%D0%A6%D0%92%D0%9A) стек называется магазином — по аналогии с [магазином в огнестрельном оружии](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D0%B3%D0%B0%D0%B7%D0%B8%D0%BD_(%D1%87%D0%B0%D1%81%D1%82%D1%8C_%D0%BE%D1%80%D1%83%D0%B6%D0%B8%D1%8F)) (стрельба начнётся с патрона, заряженного последним). Возможны три операции со стеком: добавление элемента (push), удаление элемента (pop) и чтение головного элемента .

При проталкивании (push) добавляется новый элемент, указывающий на элемент, бывший до этого головой. Новый элемент теперь становится головным.

При удалении элемента (pop) убирается первый, а головным становится тот, на который был указатель у этого объекта (следующий элемент). При этом значение убранного элемента возвращается.[2]

## **1.2 Постановка задачи**

Формирование стека, содержащего целые положительные числа, и его преобразова­ния путем удаления из него всех четных чисел (в процессе преобразования стека его элементы в оперативной памяти перемещаться не должны) ;

## **1.3 Описание программы и её решение**

### **C:\Users\Acer\Desktop\оаип1\курсовые19\Курсовая_Воробей\стек принт.PNGC:\Users\Acer\Desktop\оаип1\курсовые19\Курсовая_Воробей\стек пуш.PNGC:\Users\Acer\Desktop\оаип1\курсовые19\Курсовая_Воробей\стек маин.PNG1.3.1 Блок-схемы алгоритмов**

Рисунок 1.3-Блок-схема функции s\_print

Рисунок 1.2-Блок-схема функции s\_push

Рисунок 1.1-Блок-схема функции main

Рисунок 1.4-Блок-схема функции s\_delete\_key

### **C:\Users\Acer\Desktop\оаип1\курсовые19\Курсовая_Воробей\стек удаление.png1.3.2 Словесное описание работы программы**

Организуем структуру для стека:

struct comp { //Структура с именем comp

int Data; //Кикие то данные

comp \*next;//Указатель типа comp на следующий эелемент

};

Для выполнения поставленной задачи необходимо создать несколько функций, каждая из которых будет выполнять некий функционал. Все необхо­димое передается функциям через параметры.

Функция для добавления элементов в стек:

void s\_push(comp \*\*top, int D) {

comp \*q;

q = new comp(); //выделяем память для нового элемента

q->Data = D; //Записываем D в Data элемента

if (\*top == nullptr) { //Если вершины нет, тоесть стек пустой

\*top = q; //вершиной стека будет новый элемент

}

else{

q->next = \*top; //Проводим связь от нового элемента, к вершине.

\*top = q; //Пишем, что вершиной теперь является новый элемент

}

}

Функция для вывода элементов стека на экран:

void s\_print(comp \*top) { //принимает указатель на вершину стека

comp \*q = top; //устанавливаем q на вершину

while (q) { //пока q не пустой (while(q) эквивалентно while(q != NULL))

printf\_s("%i", q->Data);//выводим на экран данные ячейки стека

q = q->next;//после того как вывели передвигаем q на следующий элемент(ячейку)

}

}

Функция ввода принимает указатель на вершину стека и число которое будет записано в поле данных. Выделяется память под новый элемент, заполняется поле данных. Далее проверка на существование вершины стека, если вершины еще нет, то новый элемент становится вершиной , иначе проводим связь от нового элемента к вершине и присваиваем ей новый элемент, теперь новый элемент является вершиной.

Функция вывода так же принимает указатель на вершину стека. Объявляется новый указатель на вершину . Запускаем цикл ,который работает пока новый указатель не станет равным NULL. В каждой итерации мы выводим содержимое поля данных элементов стека и передвигаем указатель на следующий элемент.

Эти функции можно применить в любой программе, где требуется стек, просто изменив поля, составляющие его информационную часть, и соответствующие па­раметры.

Функция , которая удаляет четные элементы из стека:

void s\_delete\_key(comp \*\*top) {//функция которая принимает вершину top

int f = 0;

comp \*q = \*top; //создаем указатель типа comp и приравниваем его на вершину стека

comp \*prev = NULL;//создаем указатель на предыдуший элемент, с начала он будет пустым

while (q != NULL) {//пока указатель q не путой, мы его будем проверять

f = 0;

if (q->Data%2 == 0) {//проверка на четность

if (q == \*top) {//если такой указатель равен вершине

\*top = q->next;//передвигаем вершину на следующий элемент

delete []q;//очищаем ячейку

q = \*top;

f = 1;

}

else{

prev->next = q->next;//Проводим связь от предыдущего к следующему

delete []q;//очищаем ячейку

f = 1;

q = prev;

}

}

if (!f){

prev = q; //запоминаем текущую ячейку как предыдущую

q = q->next;

}//перемещаем указатель q на следующий элемент

}

}

Функция удаления четных элементов принимает указатель на вершину стека. Если находится четный элемент, то проводится связь от предыдущего элемента к следующему, а память текущего очищается.

Функция main(){…}

void main() {

comp \*top = nullptr; //в начале программы у нас нет очереди, соответственно вершины нет

for (int i = 1; i < 10; i++){

s\_push(&top, i);

}

s\_print(top);//выводим

s\_delete\_key(&top); //Затем удаляем

printf\_s("\n");//переводим на новую строку

s\_print(top);//выводим

printf\_s("\n");//переводим на новую строку

system("pause");//ставим на паузу

}

В главной функции в цикле несколько раз вызывается функция записи в стек , потом функция вывода элементов стека, затем функция удаления и снова демонстрируется уже преобразованный стек. При запуске программы отображается список элементов исходного стека и список элементов стека после удаления четных элементов.

## **1.4Тестирование**

Запустим программу.

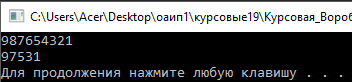


Рисунок 1.5-Результат работы программы

Программа показала исходный стек, а затем преобразованный путём удаления четных элементов. Тестирование программы окончено.

# 2 ЛИНЕЙНЫЕ СПИСКИ

**2.1 Теоретический вопрос**

Каждый узел двунаправленного (двусвязного) линейного списка (ДЛС) содержит два поля указателей — на следующий и на предыдущий узлы. Указатель на предыдущий узел корня списка содержит нулевое значение. Указатель на следующий узел последнего узла также содержит нулевое значение.

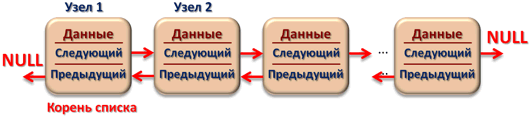


Рисунок 2.1- Двусвязный список

Основные действия, производимые над узлами ДЛС:

Инициализация списка

Добавление узла в список

Удаление узла из списка

Удаление корня списка

Вывод элементов списка

Вывод элементов списка в обратном порядке

Инициализация списка предназначена для создания корневого узла списка, у которого поля указателей на следующий и предыдущий узлы содержат нулевое значение.

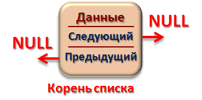


Рисунок 2.2- корень двусвязного списка

Удаление узла ДЛС.В качестве аргументов функции удаления узла ДЛС передается указатель на удаляемый узел. Поскольку узел списка имеет поле указателя на предыдущий узел, нет необходимости передавать указатель на корень списка[3]  
Удаление узла может быть представлено следующей схемой:

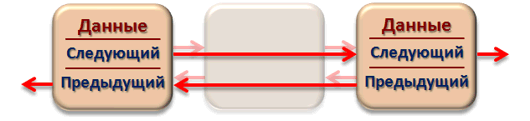


Рисунок 2.3-Удаление элемента

**2.2 Постановка задания**

Написать программу, моделирующую заполнение гибкого магнитного диска.

Общий объем памяти на диске 360 Кбайт. Файлы имеют произвольную длину от 18 байт до 32 Кбайт. В процессе работы файлы либо записываются на диск, либо удаляются с него.

В начале работы файлы записываются подряд друг за другом. После удаления фай­ла на диске образуется свободный участок памяти, и вновь записываемый файл либо размещается на свободном участке, либо, если файл не помещается в свобод­ный участок, размещается после последнего записанного файла.

В случае, когда файл превосходит длину самого большого свободного участка, выдается аварийное сообщение. Требование на запись или удаление файла задает­ся в командной строке, которая содержит имя файла, его длину в байтах, признак записи или удаления. Программа должна выдавать по запросу сведения о занятых и свободных участках памяти на диске.

Указание: следует создать список занятых участков и список свободных участков памяти на диске.

## **2.3 Описание программы и её решения**

### **2.3.1Лок-схемы алгоритмов**

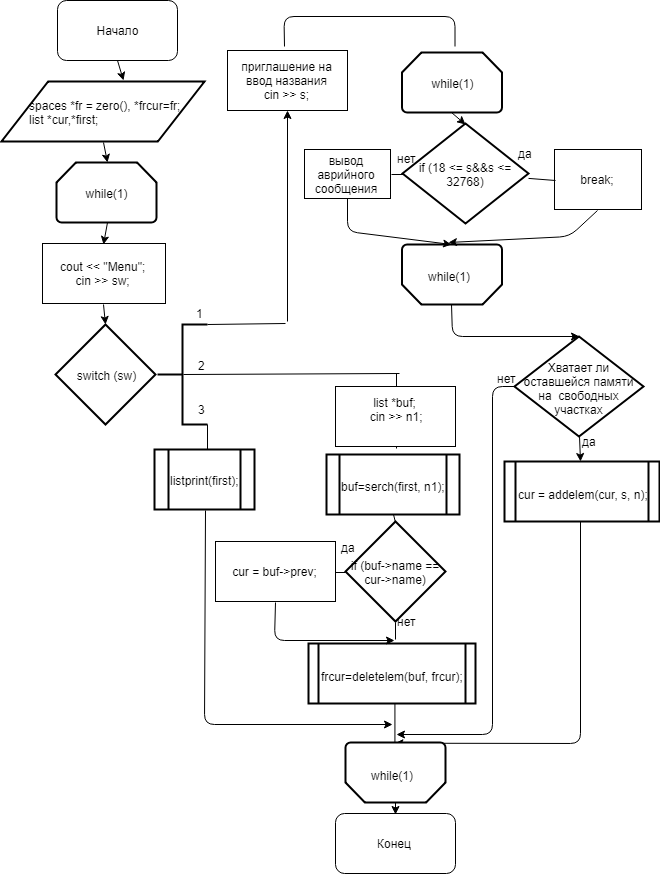
****

Рисунок 2.4-Блок-схема главной функции

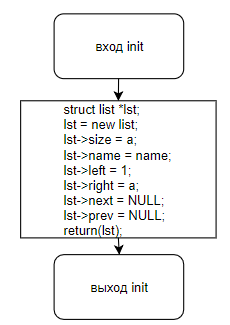
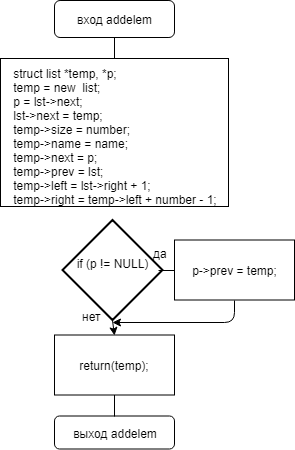
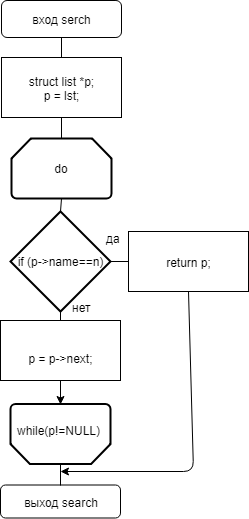


Рисунок 2.8-Блок-схема функции search

Рисунок 2.7-Блок-схема функции listprint

Рисунок 2.6-Блок-схема функции addelem

Рисунок 2.5-Блок-схема функции init

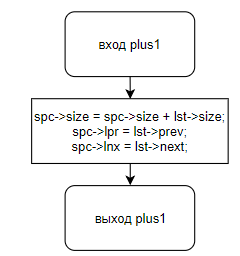
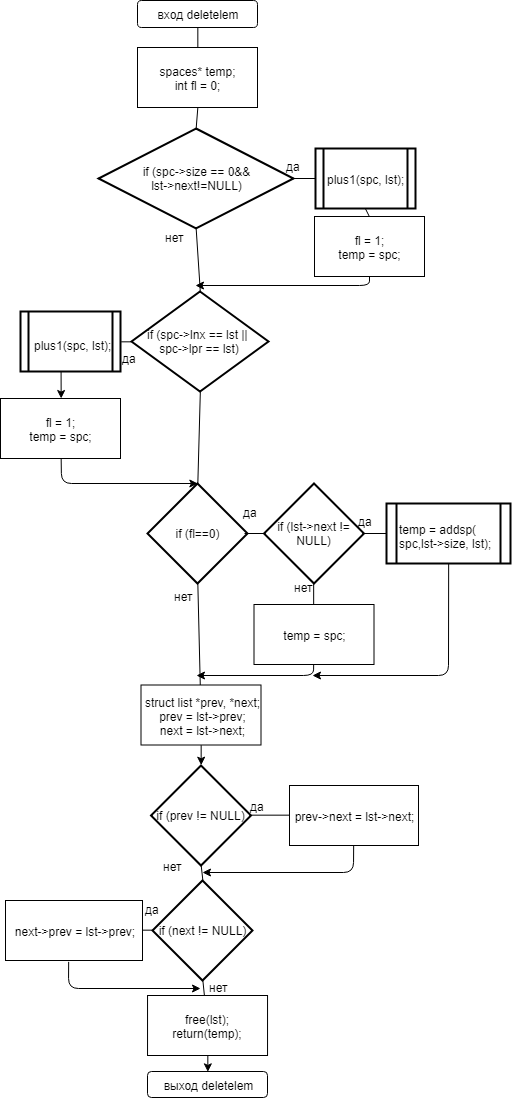


Рисунок 2.10-Блок-схема функции plus1

Рисунок 2.9-Блок-схема функции delelem

### **2.3.2 Словесное описание работы программы**

Организуем структуру для записываемых файлов:

struct list

{

string name;

int size, left, right; // поле данных

struct list \*next; // указатель на следующий элемент

struct list \*prev; // указатель на предыдущий элемент

};

А так же организуем структуру для пустых мест памяти диска, возникающих после удаления файлов:

struct spaces

{

int size, left, right; // поле данных

struct spaces \*next; // указатель на следующий элемент

struct spaces \*prev; // указатель на предыдущий элемент

struct list \*lpr,\*lnx;

};

Для реализации поставленной задачи программа разбита на ряд функций, каждая из которых отвечает за выполнение определенной задачи.

Void main(int argc,char\* argv[])

Spaces \* zero()

Void change(spaces\* spc,list\* lst)

Viod plus1(spaces\* spc,list\* lst)

List\* addelem(list\* lst, int number, string name)

Spaces\* addsp(spaces\* lst, int size, list l)

Void listprint(list\* lst)

Space\* deletelem(list\* lst, spaces\* spc)

List\* search(list\* lst, string n)

При запуске программа просит пользователя выбрать действие: добавление файла, удаление файла, вывод всех файлов с обзором занимаемого ими места на диске.

При добавлении новой записи на экран выводится приглашение на ввод размера файла, далее программа проверяет корректность введенного размера, если размер не корректен, то выводится аварийное сообще, если же размер введен верно, то проверяется если flag(этот флаг показывает создан ли уже список или еще нет) , то проверяется достаточно ли памяти на всех свободных местах диска, если достаточно , то вызывается функция addelem. Если не flag , то вызывается init , которая создает корневой элемент списка.

Функция addelem выделяет память под новый элемент, связывает его указателями с предыдущим элементом и следующим если он есть. В поле данных записывается размер, границы занимаемой памяти и название файла.

Если пользователь выбирает удаление файла, то выводится приглашение на ввод имени файла, который следует удалить, далее вызывается функция search, которая возвращает элемент списка с таким именем. Далее вызывается функция deletelem, которая принимает удаляемый элемент и текущий элемент другого списка(spaces список пробелов в памяти). Эта функция проверяет есть ли рядом с удаляемым файлом свободное место в памяти, если есть то оно суммируется , если нет то создается новый элемент списка пробелов памяти spaces с помощью функции addsp. Функция addsp работает аналогично addelem.

Если пользыватель выбирает обзор памяти диска, то вызывается функция listprint.

## **2.4 Тестирование**

Запустим программу. Программа выводит меню.

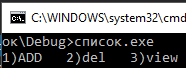


Рисунок 2.11- меню программы

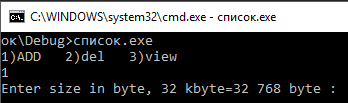


Рисунок 2.12-Запрос на ввод размера

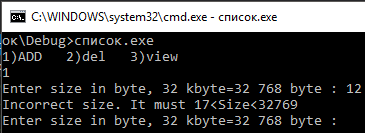


Рисунок 2.13-Демонстрация защиты от некорректного ввода

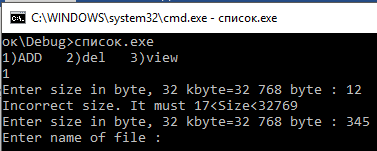


Рисунок 2.14- Приглашение на ввод названия файла

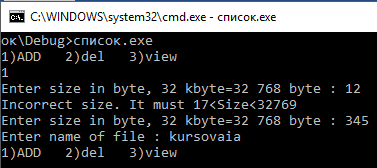


Рисунок 2.15- Демонстрация повторного вывода меню после ввода данных

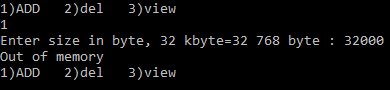


Рисунок 2.16- Демонстрация аварийного сообщения

На рисунке 2.17 видно, что после удаления двух файлов(bcng,foto) , которые расположены в памяти один за другим, образовался общий промежуток свободной памяти после 96345 байта и до 160346 байта. А так же был удален файл “hello”, после чего образовался свободный участок памяти после 224337байта и до 256378 байта.

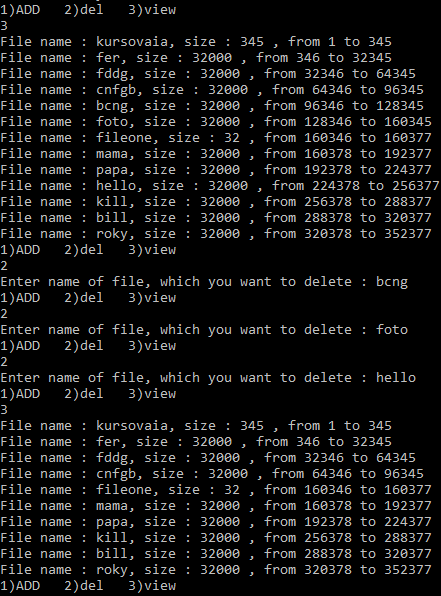


Рисунок 2.17- Демонстрация файлов на диске и объема, занимаемой памяти. Удаление и демонстрация диска после удаления

На рисунке 2.18 видно, что на место освободившееся от двух удаленных элементов, было записано четыре новых(x...x,y…y,z…z,i…i). На этом участке не занятыми остались байты от 132213 до 160345 включительно.

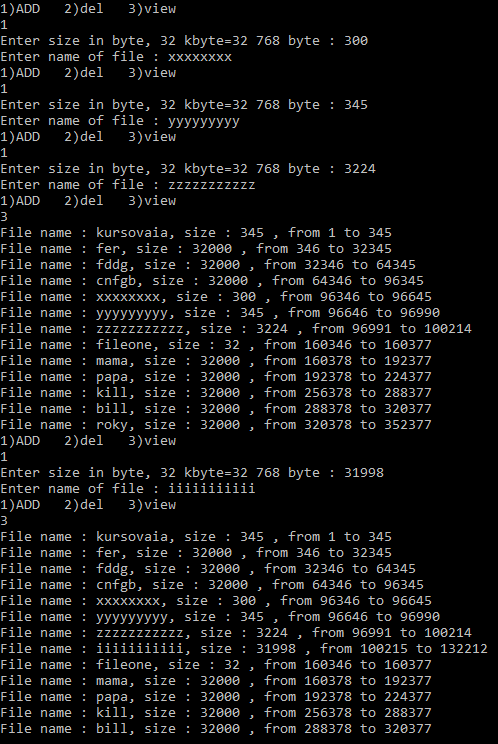


Рисунок 2.18- Запись новых элементов на свободный после удаления участок памяти

На рисунке 2.19 показан ввод элемента размером 30000 байт. На первом очищенном промежутке , куда мы записали файлы x...x,y…y,z…z,i…i осталось только 28132 свободных байт и поэтому новый файл w…w будет записан на следующий свободный промежуток памяти.

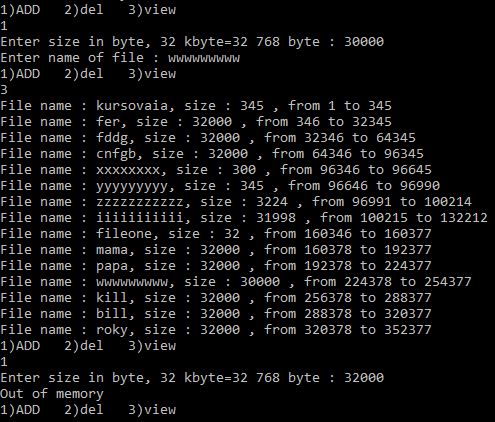


Рисунок 2.19- Демонстрация записи элемента на следующий подходящий по размеру свободный после удаления участок памяти

**ЗАКЛЮЧЕНИЕ**

Подводя итоги курсовой работы по дисциплине «Основы алгоритмизации и программирования», можно сделать выводы о том, что поставленные цели и задачи были успешно выполнены, так же были повторены основы языка программирования C++. На более глубоком уровне изучена работа со списками со стеками. Изучены сложные и нестандартные алгоритмы. Отработаны практические навыки и умения, полученные в процессе изучения базовой части дисциплины «Основы алгоритмизации и программирования». Усовершенствовали теоретические знания.

Во время выполнения курсовой работы использовались полученные знания во время изучения дисциплины «Основы алгоритмизации и программирования». В программе продемонстрированы такие навыки работы, как работа с линейным списком и работа со стеками;

# 

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ И ИСТОЧНИКОВ

# О стеке простыми словами -для студентов и просто начинающих [Электронный ресурс]. Пост от 2 ноября 2017  URL: <https://habr.com/ru/post/341586/>

# Стек. Материал из Википедии — свободной энциклопедии [Электронный ресурс].URL: [https://ru.wikipedia.org/wiki/](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B5%D0%BA)Стек

# [Структуры данных](https://prog-cpp.ru/category/struct-posts/) / Двусвязный линейный список. [Электронный ресурс]. URL: <https://prog-cpp.ru/data-dls/>

# ПРИЛОЖЕНИЕ А

**Код программы:**

#include <stdio.h>;

#include <iostream>

struct comp { //Структура с именем comp

int Data; //

comp \*next;//Указатель типа comp на следующий эелемент

};

void s\_push(comp \*\*top, int D) {

comp \*q;

q = new comp(); //выделяем память для нового элемента

q->Data = D; //Записываем D в Data элемента

if (\*top == nullptr) { //Если вершины нет, тоесть стек пустой

\*top = q; //вершиной стека будет новый элемент

}

else{

q->next = \*top;

\*top = q; //Пишем, что вершиной теперь является новый элемент

}

}

void s\_delete\_key(comp \*\*top) {//функция которая принимает вершину top

int f = 0;

comp \*q = \*top;

comp \*prev = NULL;//создаем указатель на предыдуший элемент, с начала он будет пустым

while (q != NULL) {

f = 0;

if (q->Data%2 == 0) {//проверка на четность

if (q == \*top) {

\*top = q->next;//передвигаем вершину на следующий элемент

delete []q;//очищаем ячейку

q = \*top;

f = 1;

}

else{

prev->next = q->next;

delete []q;//очищаем ячейку

f = 1;

q = prev;

}

}

if (!f){

prev = q; //запоминаем текущую ячейку как предыдущую

q = q->next;

}//перемещаем указатель q на следующий элемент

}

}

void s\_print(comp \*top) { //принимает указатель на вершину стека

comp \*q = top; //устанавливаем q на вершину

while (q) { //пока q не пустой (while(q) эквивалентно while(q != NULL))

printf\_s("%i", q->Data);//выводим на экран данные ячейки стека

q = q->next;//после того как вывели передвигаем q на следующий элемент(ячейку)

}

}

void main() {

comp \*top = nullptr;

for (int i = 1; i < 10; i++){

s\_push(&top, i);

}

s\_print(top);//выводим

s\_delete\_key(&top); //Затем удаляем

printf\_s("\n");//переводим на новую строку

s\_print(top);//выводим

printf\_s("\n");//переводим на новую строку

system("pause");//ставим на паузу

}

# ПРИЛОЖЕНИЕ Б

**Код программы:**

#include<iostream>

#include<string>

using namespace std;

struct list

{

string name;

int size, left, right; // поле данных

struct list \*next; // указатель на следующий элемент

struct list \*prev; // указатель на предыдущий элемент

};

struct spaces

{

int size, left, right; // поле данных

struct spaces \*next; // указатель на следующий элемент

struct spaces \*prev; // указатель на предыдущий элемент

struct list \*lpr,\*lnx;

};

list \* init(int a, string name) {

struct list \*lst;// выделение памяти под корень списка

lst = new list;

lst->size = a;

lst->name = name;

lst->left = 1;

lst->right = a;

lst->next = NULL; // указатель на следующий узел

lst->prev = NULL; // указатель на предыдущий узел

return(lst);

}

spaces \* zero(){

struct spaces \*spc;// выделение памяти под корень списка

spc = new spaces;

spc->next = NULL; // указатель на следующий узел

spc->prev = NULL; // указатель на предыдущий узел

spc->size = 0;

return(spc);

}

void change(spaces\* spc, list\* lst){

spc->size = spc->size - lst->size;

spc->lpr = lst;

}

void plus1(spaces\* spc,list \*lst){

spc->size = spc->size + lst->size;

spc->lpr = lst->prev;

spc->lnx = lst->next;

}

list \* addelem(list \*lst, int number, string name)

{

struct list \*temp, \*p;

temp = new list;

p = lst->next; // сохранение указателя на следующий узел//

lst->next = temp; // предыдущий узел указывает на создаваемый

temp->size = number; // сохранение поля данных добавляемого узла

temp->name = name;

temp->next = p; // созданный узел указывает на следующий узел

temp->prev = lst; // созданный узел указывает на предыдущий узел

temp->left = lst->right + 1;

temp->right = temp->left + number - 1;

if (p != NULL)

p->prev = temp;

return(temp);

}

spaces \* addsp(spaces \*lst, int size,list\*l)

{

struct spaces \*temp, \*p;

temp = new spaces;

p = lst->next; // сохранение указателя на следующий узел

lst->next = temp; // предыдущий узел указывает на создаваемый

temp->size = size; // сохранение поля данных добавляемого узла

temp->next = p; // созданный узел указывает на следующий узел

temp->prev = lst; // созданный узел указывает на предыдущий узел

temp->left = lst->right + 1;

temp->right = temp->left + size - 1;

temp->lpr = l->prev;

temp->lnx = l->next;

if (p != NULL)

p->prev = temp;

return(temp);

}

void listprint(list \*lst){

struct list \*p;

p = lst;

do {

cout << "File name : " << p->name << ", size : " << p->size << " , from " << p->left << " to " << p->right << endl;

p = p->next; // переход к следующему узлу

} while (p != NULL); // условие окончания обхода

}

spaces \* deletelem(list \*lst,spaces \*spc){

spaces\* temp;

int fl = 0;

if (spc->size == 0&&lst->next!=NULL){

plus1(spc, lst);

fl = 1;

temp = spc;

}

if (spc->lnx == lst || spc->lpr == lst){

plus1(spc, lst);

fl = 1;

temp = spc;

}

if (fl==0)

{

if (lst->next != NULL)

temp = addsp(spc, lst->size, lst);

else

temp = spc;

}

struct list \*prev, \*next;

prev = lst->prev; // узел, предшествующий lst

next = lst->next; // узел, следующий за lst

if (prev != NULL)

prev->next = lst->next; // переставляем указатель

if (next != NULL)

next->prev = lst->prev; // переставляем указатель

free(lst); // освобождаем память удаляемого элемента

return(temp);

}

list \* serch(list\*lst,string n){

struct list \*p;

p = lst;

do {

if (p->name==n)

return p;

p = p->next; // переход к следующему узлу

} while (p != NULL); // условие окончания обхода

}

void main(int argc, char\* argv[]) {

string n ,n1;

spaces \*fr = zero(), \*frcur=fr;

list \*cur,\*first;

struct list \*p,\*p1;

int sw,flag=0,i=0,fl=1,f=1;

while (1){

cout << "1)ADD\t2)del\t3)view\n";

cin >> sw;

switch (sw)

{

case 1:

int s;

while (1){

cout << "Enter size in byte, 32 kbyte=32 768 byte : "; cin >> s;

if (18 <= s&&s <= 32768)

break;

else

cout << "Incorrect size. It must 17<Size<32769\n";

}

if (flag){

if (368640 - cur->right - s < 0){

p = p1=first;

p1 = p1->next;

do {

f = 1;

if (p1->left - p->right >= s){

f = 0;

break;

}

p = p->next; // переход к следующему узлу

p1 = p1->next;

} while (p1->next != NULL); // условие окончания обхода

if (f == 1){

cout << "Out of memory\n";

continue;

}

}

}

cout << "Enter name of file : "; cin >> n;

if (!flag){

first = init(s,n);

cur = first;

}

else{

fl = 1;

struct spaces \*p;

p = fr;

do {

if (p->size >= s){

change(p, addelem(p->lpr, s, n));

fl = 0;

break;

}

p = p->next; // переход к следующему узлу

} while (p != NULL); // условие окончаобхода

if (fl){

cur = addelem(cur, s, n);

}

}

i++;

flag = 1;

break;

case 2:

list \*buf;

cout << "Enter name of file, which you want to delete : "; cin >> n1;

buf=serch(first, n1);

if (buf->name == cur->name)

cur = buf->prev;

frcur=deletelem(buf, frcur);

break;

case 3:

listprint(first);

break;

}

}

system("pause");

}