## Лабораторная работа №14

## Линейные списки

***Цель лабораторной работы:*** *изучение способов создания и принципов использования односвязных линейных списков; изучение стандартных средств языка C/C++ для работы с динамической памятью; совершенствование навыков модульного программирования на языке C/C++ при решении задач обработки линейных списков; изучение способов разработки многофайловых проектов.*

***Задание на программирование:*** *используя технологию процедурного программирования разработать программу обработки односвязных линейных списков с числом элементов в списке* *не менее десяти в соответствии с индивидуальным заданием.*

***Порядок выполнения работы:***

1. Получить у преподавателя индивидуальное задание.

2. Провести структуризацию задачи. С этой целью выделить подзадачи, которые будут реализованы в виде отдельных функций. При этом запрещается совмещать в одной функции решение нескольких подзадач.

3. Составить описание процесса решения задачи.

4. Построить схему алгоритма решения задачи (функции *main*()) с использованием функций создания, просмотра, обработки списка, удаления списка из динамической памяти.

5. Обосновать перечень и типы параметров всех функций.

6. Построить схему алгоритма функции обработки списка.

7. Создать многофайловый проект на языке *C*/*C*++.

8. Проверить и продемонстрировать преподавателю работу программы на полном наборе тестов. Обеспечить одновременный показ на экране исходного и результирующего списков.

9. Выходные данные должны выводиться на экран с пояснениями. Операторы вывода результатов работы должны находиться либо в функции *main*(), либо в специальной функции вывода (например, преобразованного списка), вызов которой осуществляется из функции *main*().

10. Оформить отчет о лабораторной работе в составе: постановка задачи, описание процесса решения, схемы алгоритмов функции *main*(0 и функции обработки списка, текст всех модулей проекта, контрольные примеры (скриншоты).

11. Текст программы в отчете не должен представлять из себя скриншот.

12. Скриншоты тестов должны легко читаться. Все их неинформативные части должны быть удалены.

***Варианты индивидуальных заданий***

**1.**

В непустом списке найти первый элемент с минимальным значением и удвоить его значение.

**2.**

В непустом списке найти максимальное и минимальное значения его элементов. Заменить значение первого элемента списка суммой найденных значений.

**3.**

В непустом списке найти количество элементов, лежащих между первым элементом с максимальным значением и первым элементом с минимальным значением.

**4.**

В непустом списке обнулить значения элементов, лежащих перед первым элементом с максимальным значением.

**5.**

В непустом списке найти последний элемент с максимальным значением и увеличить в два раза его значение.

**6.**

В непустом списке обнулить все отрицательные значения элементов, лежащих после первого элемента с максимальным значением.

**7.**

В непустом списке обнулить значения элементов, лежащих между первым элементом с максимальным значением и последним элементом с минимальным значением.

**8.**

В непустом списке обнулить значения элементов, лежащих после первого элемента с минимальным значением.

**9.**

В непустом списке найти максимальную сумму значений двух любых его элементов. Заменить значение первого элемента списка найденной суммой.

**10.**

В непустом списке найти максимальную сумму значений двух любых его элементов с отрицательным значением. Заменить значение последнего отрицательного элемента списка найденной суммой.

**11.**

В списке *L* из каждой группы подряд идущих элементов с равными значениями оставить только один.

**12.**

Найти первый элемент непустого списка с максимальным значением и изменить знак его значения на противоположный.

**13.**

Перевернуть исходный список *L*, то есть изменить ссылки в этом списке так, чтобы его элементы оказались расположенными в обратном порядке. Дополнительный список не использовать.

**14.**

В непустом списке найти минимальную сумму значений двух любых его элементов с положительным значением. Заменить значение первого положительного элемента списка найденной суммой.

**15.**

Проверить, есть ли в списке *L* хотя бы два элемента с одинаковыми значениями.

**16.**

В непустом списке найти наибольшее значение среди элементов с отрицательным значением и наименьшее значение среди элементов с положительным значением. Заменить значение последнего положительного элемента списка суммой найденных значений.

**17.**

Перенести в конец списка два его первых элемента таким образом, чтобы последним в списке оказался тот из них, значение которого меньше значения второго.

**18.**

Вставить в список *L* за последним вхождением элемента со значением *E* все элементы списка *L*1, если элемент со значением *E* входит в *L*.

**19.**

Сформировать список *L*, включив в него по одному разу элементы, которые входят хотя бы в один из списков *L*1 и *L*2.

**20.**

Сформировать список *L*, включив в него по одному разу элементы, которые входят одновременно в оба списка *L*1 и *L*2.

**21.**

Сформировать список *L*, включив в него по одному разу элементы, которые входят в список *L*1, но не входят в список *L*2.

**22.**

Сформировать список *L*, включив в него по одному разу элементы, которые входят в один из списков *L*1 и *L*2, но в то же время не входят в другой из них.

**23.**

Объединить два упорядоченных списка *L*1 и *L*2 в один упорядоченный список, построив новый список *L*.

**24.**

Объединить два упорядоченных списка *L*1 и *L*2 в один упорядоченный список *L*1, меняя соответствующим образом ссылки в *L*1 и *L*2.

**25.**

Найти среднее арифметическое значений элементов непустого списка.

**26.**

Подсчитать количество слов списка, которые начинаются и оканчиваются одной и той же литерой.

**27.**

Вставить в непустой список новый элемент *E* после первого элемента списка с наибольшим значением.

**28.**

Поменять местами первый и последний элемент списка.

**29.**

Проверить, упорядочены ли элементы списка по алфавиту.

**30.**

Удалить из списка последний отрицательный элемент, если такой есть.

**31.**

Заменить в списке *L* последнее вхождение элемента со значением *E*1 на *E*2.

**32.**

Перенести в конец списка его первый отрицательный элемент.

**33.**

Удалить из списка второй элемент с наибольшим значением, если такой есть.

**34.**

Подсчитать число вхождений элемента со значением *E* в список *L* и умножить на это число значение второго положительного элемента списка.

**35.**

Удалить из списка *L* последнее вхождение элемента *E*, если такое есть.

**36.**

Перенести в начало списка его последний отрицательный элемент.

**37.**

Подсчитать количество слов списка, которые совпадают с последним словом.

**38.**

Найти сумму значений последнего и предпоследнего отрицательных элементов списка.

**39.**

Подсчитать количество слов списка, которые оканчиваются той же литерой, что и следующее в списке слово.

**40.**

В непустом списке найти первый элемент с максимальным значением и последний элемент с минимальным значением и поменять их местами.

**41.**

По списку *L* построить два новых списка *L*1 и *L*2: первый из элементов с положительными значениями, а второй из остальных элементов списка *L*.

**42.**

Вставить в список *L* новый элемент со значением *E*1 за последним элементом с заданным значением *E*, если элемент со значением *E* входит в *L*.

**43.**

Вставить в список *L* новый элемент со значением *E*1 перед последним вхождением элемента с заданным значением *E*, если элемент со значением *E* входит в *L*.

**44.**

Вставить в непустой список *L* перед его последним отрицательным элементом пару новых элементов со значениями *E*1 и *E*2.

**45.**

Удвоить каждое вхождение элемента со значением *E* в списке *L*.

**46.**

Удалить из списка *L* за последним вхождением элемента со значением *E* один элемент, если он есть.

**47.**

Вставить в непустой список *L*, элементы которого изначально упорядочены по не убыванию их значений, новый элемент со значением *E* так, чтобы сохранить упорядоченность элементов списка.

**48.**

Удалить из списка *L* последнее вхождение элемента со значением *Е*.

**49.**

Удалить из списка *L* все элементы с отрицательными значениями.

**50.**

Оставить в списке *L* только первые вхождения одинаковых элементов.

***Пример 1 решения задачи на обработку линейного списка для задания вида:***

Создать линейный список *L* с произвольной информационной частью и содержащий не менее трех элементов. Удалить из созданного списка все элементы, у которых соседи (предыдущий и последующий элементы) имеют одинаковые значения информационных частей. Реализовать многофайловый проект.

***Описание процесса решения***

В разрабатываемый проект будут входить три файла:

- заголовочный файл *modSp*.*h*, содержащий определения новых типов и объявления функций;

- исходный файл *modSp*.*cpp*, содержащий реализацию набора функций для обработки списка;

- основной файл *spis*.*cpp* с программой (функция *main*()).

При решении задачи, а значит в тексте проекта, кроме функции *main*() необходимо реализовать следующие функции:

- функцию инициализации списка;

- функцию добавления нового элемента в конец списка;

- функцию обработки списка в соответствии с заданием;

- функцию просмотра содержимого списка;

- функцию удаления списка из динамической памяти.

Конкретный тип значений информационной части элементов списка в задании не указан, поэтому для получения легко модернизируемой программы с целью изменения типа обрабатываемых данных определяем новый тип с использованием *typedef*:

- тип значений информационной части элементов списка (*telem*). На этапах разработки и тестирования программы это будет тип *char*.

Определяем новый тип - структуру *list*, которая будет описывать элементы списка. У неё два поля: информационное поле типа *telem* и поле адреса - указатель на следующий элемент списка (*list*\*).

Эти определения новых типов помещаем в заголовочный файл *modSp*.*h*. Сюда же будут помещены объявления функций, необходимых для решения задачи.

Функцию инициализации списка называем *init\_spis*(*list*\*\*,*list*\*\*). Тип возвращаемого функцией значения - *void*. В задании не сказано, как должен формироваться список: подключением новых элементов к голове или к хвосту списка. Выбираем способ формирования с подключением новых элементов к концу списка. Поэтому функция инициализации будет иметь два параметра типа “указатель на указатель на тип *list*”. Первый хранит адрес адреса головного элемента списка, а второй – адрес адреса концевого элемента. Под инициализацией списка будем понимать присваивание адресам головного и концевого элементов списка значения *NULL* (т.е. список пуст).

Функцию добавления нового элемента в конец списка называем *add*\_*spis*(*telem*,*list*\*\*,*list*\*\*). Тип возвращаемого функцией значения - *void*. У неё три параметра: значение информационной части нового элемента списка типа *telem* и два параметра типа “указатель на указатель на тип *list*” (адреса головного и концевого элементов списка будут меняться при подключении нового элемента и передаются через указатели).

Функцию обработки списка называем *obrabotka*(*list*\*). В соответствии с условиями задачи адрес головного и концевого элементов при обработке измениться не могут. Тип возвращаемого функцией значения - *void*. У неё один параметр типа “указатель на тип *list*”. В функцию передаётся значение адреса головного элемента списка.

Функцию просмотра содержимого списка называем *view\_spis*(*list*\*). Тип возвращаемого функцией значения - *void*. У неё один параметр типа “указатель на тип *list*”. В функцию передаётся значение адреса головного элемента списка. Под просмотром содержимого списка здесь понимается вывод пользователю на экран значений информационных частей всех элементов списка.

Функцию удаления списка из динамической памяти называем *udal\_spis*(*list*\*\*,*list*\*\*). Тип возвращаемого функцией значения - *void*. У неё два параметра типа “указатель на указатель на тип *list*” (адреса головного и концевого элементов списка будут меняться при удалении списка из динамической памяти), при этом инициализация списка сохраняется.

На этом формирование заголовочного файла *modSp*.*h* заканчивается.

**В исходный файл *modSp*.*cpp*** помещаем реализацию набора функций для работы со списком.

Остановимся подробнее на разработке функции *obrabotka*() (реализации процесса решения задачи).

Согласно заданию на разработку необходимо из созданного списка удалить все элементы, у которых значения информационных полей соседей совпадают. Для того чтобы удалить элемент из списка, нужно, прежде всего, установить указатель на предыдущий элемент. По условию задачи могут удаляться элементы, у которых есть соседи. Это элементы со второго до предпоследнего элемента списка. Поэтому для решения задачи надо организовать цикл проверки необходимости удаления элементов, устанавливая указатель *tec* на предыдущий по отношению к ним элемент списка, т.е. на элементы с первого (*tec*=*first*) до предпредпоследнего элемента списка (*while*(*tec*->*link*->*link*)).

Если при этом совпадают значения информационных частей соседей

(*tec*->*data*==*tec*->*link*->*link*->*data*),

то в ссылочное поле текущего элемента записывается адрес элемента из ссылочного поля последующего

(*tec*->*link*=*tec*->*link*->*link*),

а сам последующий элемент удаляется операцией *delete*.

После удаления элемента из списка у нового соседа может быть такое же информационное поле. Поэтому, не меняя текущего элемента необходимо повторять все действия до тех пор, пока у элемента, следующего за текущим элементом, не будет одинаковых «соседей»:

*while*(*tec*->*data*==*tec*->*link*->*link*->*data*)).

Теперь надо перейти на следующий элемент списка (*tec*=*tec*->*link*), и повторить все действия для него.

После завершения внешнего цикла из-за удаления отдельных элементов могут опять появиться элементы, у которых их соседи имеют одинаковые значения информационных частей. Это значит, что все действия надо опять повторять с начала списка, если хотя бы один элемент был удален из списка.

Вводим переменную *pr*, выполняющую роль флажка (признака) необходимости выполнения описанных действий. В исходном состоянии её значение *pr*=1 (действия надо выполнить хотя бы один раз). Организуем цикл повторения последовательности проверок (*while*(*pr*)). Предполагаем, что в списке отсутствуют элементы, которые надо удалять (*pr*=0). Выполняем последовательность проверок и удалений, и если хотя бы один элемент удалялся, делаем значение флажка *pr*=1. Выход из внешнего цикла при значении *pr*=0.

Схема алгоритма функции *obrabotka*() приведена на рисунке.

**В основной файл *spis*.*cpp*** помещаем функцию *main*(). Исполнение программы (функции *main*()) начинается с объявления переменных. Это указатели на первый и последний элементы списка *first* и *end* и переменная *ch* типа *telem*.

Инициализируем исходный список. Для этого в функцию *main*() помещаем вызов функции *init\_spis*(), параметрами которой являются адреса указателей на первый и последний элементы списка (&*first*, &*end*).

Приступаем к созданию линейного списка. Поскольку линейный список – это динамическая структура, пользователю выводится подсказка с указанием признака окончания ввода. Если у элементов списка символьная информационная часть, то для примера оговариваем, что признаком окончания ввода значений является символ точка ‘.’. Организуем цикл *while*(*ch* != '.').

***Схема алгоритма функции обработки (obrabotka)***

*obrabotka*(*first*)

*pr*=1

нет

*pr==1*

да

*pr*=0

*tec*=*first*

*tec*->*link*->*link*!=0

нет

да

*tec*->*data==tec*->*link*->*link*->*data*

нет

да

*vr*=*tec*->*link*

*tec*->*link*= *tec*->*link*->*link*

*delete vr*

*pr*=1

*tec*=*tec*->*link*

*return*

В тело этого цикла помещаем вызов функции добавления нового элемента в список *add*\_*spis*(), передавая ей в качестве параметров значение очередного считанного символа *ch* и адреса указателей на первый и последний элементы списка (&*first*, &*end*).

После выхода из цикла проверяем факт создания списка, сравнивая значение указателя на первый элемент списка со значением *NULL*. Если список не создан, выполнение программы заканчивается (*return* 0).

Если список создан, выводим значения его элементов, вызывая функцию *view\_spis*(), параметром которой является значение указателя на первый элемент созданного списка *first*.

Проверка создания списка выполнена. Приступаем к обработке созданного списка. Вызываем функцию *obrabotka*(), передавая ей в качестве параметра значение указателя на первый элемент списка *first*.

После обработки списка выводим значения его элементов, вызывая функцию *view\_spis*(), параметром которой является значение указателя на первый элемент обработанного списка *first*.

Программа завершается. Удаляем список из динамической памяти, вызывая функцию *udal\_spis*(), передавая ей в качестве параметров значения адресов указателей на первый и последний элементы списка (&*first*, &*end*).

Анализируем результаты работы проекта и делаем выводы.

Схема алгоритма функции *main*() приведена на рисунке.

***Схема алгоритма функции main()***

начало

*init\_spis*(&*first*,&*end*)

ввод

*ch*

нет

*ch*!=’.’

да

*add\_spis*(*ch*,&*first*,&*end*)

ввод

*ch*

нет

*first*!=0

да

*view\_spis*(*first*)

*obrabotka*(*first*)

*view\_spis*(*first*)

*udal\_spis*(&*first*,&*end*)

конец

***Текст программ проекта (пример 1)***

***Заголовочный файл modSp.h***

/\*Файл modSp.h

Заголовочный файл

Содержит определения новых типов и объявления функций\*/

//Определения типов

typedef char telem ; //определение типа информационного поля

struct list //определение типа элемента списка

{telem data ; //информационное поле

list \*link ; //поле адреса

} ;

//Объявления (прототипы) функций

void init\_spis(list\*\*,list\*\*) ; //инициализация списка

void add\_spis(telem,list\*\*,list\*\*) ; //добавление элемента

void obrabotka(list\*) ; //обработка списка

void view\_spis(list\*) ; //просмотр списка

void udal\_spis(list \*\*,list\*\*) //освобождение памяти

***Основной файл spis.cpp***

/\*Основной файл spis.cpp

Создать линейный список L.

Удалить из списка L, содержащего не менее 2-х элементов,

все элементы, у которых соседи (предыдущий и последующий элементы)

имеют одинаковые значения информационных полей.

Многофайловый проект\*/

#include <iostream.h>

#include "modSp.h"

using namespace std ;

//main\_beg\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//основная программа

int main()

{list \*first, \*end ;

telem ch ;

setlocale(LC\_ALL,”Russian”) ;

init\_spis(&first, &end) ;

//создание исходного списка

cout << "\n Удалить из списка L, содержащего не менее 2-х элементов," <<

"\n все элементы, у которых соседи (предыдущий и последующий элементы)"

<< "\n имеют одинаковые значения информационных полей" << endl ;

cout << " Введите элементы списка (символы) сплошной строкой;"

<< " в конце - точка:" << endl ;

cin >> ch ;

while(ch != '.')

{add\_spis(ch, &first, &end) ;

cin >> ch ;

}

//вывод исходного списка

if(first && first->link && first->link->link) //список создан правильно

{cout << " Исходный список:" << endl ;

view\_spis(first) ;

//обработка списка

obrabotka(first);

cout << "\nРезультат получен:\n";

view\_spis(first);

//освобождение памяти

udal\_spis(&first, &end) ;

}

else cout << "Список не создан или в нем меньше 3-х элементов" ;

return 1;

}

***Исходный файл modSp.cpp***

/\*Файл modSp.cpp

Исходный файл

Реализация набора функций для обработки списка\*/

#include <iostream.h>

#include "modSp.h"

using namespace std ;

setlocale(LC\_ALL,”Russian”) ;

//obrabotka\_beg\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//Обработка списка

void obrabotka(list \*first)

{list \*tec,

\*vr ;

int pr = 1 ; //признак: хотя бы одна проверка нужна

while(pr) //пока проверка нужна

{pr = 0 ;

tec = first ; //указатель устанавливаем на голову списка

while(tec -> link -> link) /\*пока не прошли предпредпоследний

элемент списка (у предпоследнего есть оба соседа)\*/

{while(tec->data == tec->link->link->data)/\*пока "соседи" совпадают\*/

{vr = tec->link ; //запоминаем адрес исключаемого

tec->link = tec->link->link ; //исключаем из списка

delete vr ; //освобождаем память

pr = 1 ; //нужно снова проверять с начала списка

}

tec = tec->link ;

}

}

return ;

}

//obrabotka\_end\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//view\_spis\_beg\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//Просмотр содержимого списка

void view\_spis(list \*tec)

{do

{cout << tec->data ;

tec = tec->link ;

}

while(tec) ;

return ;

}

//view\_spis\_end\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//add\_spis\_beg\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//Добавление нового элемента в конец списка

void add\_spis(telem ch, list \*\*first, list \*\*end)

{list \*nov = new list ; //выделяем память под элемент списка

nov->data = ch ; //заполняем информационную часть

nov->link = 0 ; //в ссылочную - NULL

if (\*first) //если список не пуст

(\*end)->link = nov ;

else \*first = nov ; //если список пока пуст

\*end = nov ; //новый элемент встал в конец списка

return ;

}

//add\_spis\_end\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//init\_spis\_beg\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//Инициализация списка

void init\_spis(list \*\*first, list \*\*end)

{\*first = \*end = 0 ; //исходный список пуст

return ;

}

//init\_spis\_end\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//udal\_spis\_beg\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//Освобождение динамической памяти

void udal\_spis(list \*\*first, list \*\*end)

{list \*tec ;

while(\*first) //начинаем с первого элемента

{tec = \*first ; //запоминаем его адрес

\*first = (\*first)->link ; /\*удаляем из списка и переходим

к следующему элементу\*/

delete tec ; //освобождаем память

}

\*end = 0 ; /\*когда удален весь список

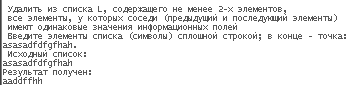
адрес последнего равен NULL\*/

return ;

}

//udal\_spis\_end\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Скриншот результатов выполнения программы (пример 1)***



***Пример 2 решения задачи на обработку линейного списка для задания вида:***

Создать линейный список с информационной частью в виде целых чисел. Разбить исходный список на два списка - из четных и нечетных значений. Реализовать многофайловый проект.

***Описание процесса решения***

В разрабатываемый проект будут входить три файла:

- заголовочный файл *modSp*1.*h*, содержащий определения новых типов и объявления функций;

- исходный файл *modSp*1.*cpp*, содержащий реализацию набора функций для обработки списка;

- основной файл *spis*1.*cpp* с программой (функция *main*()).

При решении задачи, а значит в тексте проекта, кроме функции *main*() необходимо реализовать следующие функции:

- функцию инициализации списка;

- функцию добавления нового элемента в конец списка;

- функцию обработки списка в соответствии с заданием;

- функцию просмотра содержимого списка;

- функцию удаления списка из динамической памяти.

Конкретный тип значений информационной части элементов списка в задании указан. Для унификации примеров решения задач определяем новый тип с использованием *typedef*:

- тип значений информационной части элементов списка (*telem*). По условию задачи ему будет соответствовать тип *int*.

Определяем новый тип - структуру *list*, которая будет описывать элементы списка. У неё два поля: информационное поле типа *telem* и поле адреса - указатель на следующий элемент списка (*list*\*).

Эти определения новых типов помещаем в заголовочный файл *modSp*1.*h*. Сюда же помещаем объявления функций, необходимых для решения задачи.

Функции инициализации списка *void init\_spis*(*list*\*\*, *list*\*\*), добавления нового элемента в конец списка *void add\_spis*(*telem*, *list*\*\*, *list*\*\*), просмотра содержимого списка *void view\_spis*(*list*\*) и функцию удаления списка из динамической памяти *void udal\_spis*(*list*\*\*, *list*\*\*) заимствуем из предыдущего проекта и здесь рассматривать не будем.

Функцию обработки списка называем *obrabotka*(*list*\*, *list*\*\*, *list*\*\*, *list*\*\*, *list*\*\*). Тип возвращаемого функцией значения - *void*. У неё один параметр типа “указатель на тип *list*” (исходный список при обработке не изменится) и четыре параметра типа “указатель на указатель на тип *list*” (адреса головного и концевого элементов результирующих списков, которые будут меняться при формировании этих списков). Адрес головного элемента исходного списка передаётся в функцию по значению. Адреса головного и концевого элементов результирующих списков передаются в функцию через указатели.

На этом формирование заголовочного файла *modSp*1.*h* заканчивается.

**В исходный файл *modSp*1.*cpp*** помещаем реализацию набора функций для работы со списком.

Остановимся подробнее на разработке функции *obrabotka*() (реализации процесса решения задачи).

Согласно заданию на разработку необходимо из созданного списка с информационной частью в виде целых чисел создать два новых списка: один из чётных значений информационных частей, а второй - из нечётных значений.

Поэтому для решения задачи надо организовать цикл проверки чётности значений информационных частей элементов списка. Если значение чётное – добавляем его к концу одного результирующего списка, если нечётное – к концу другого списка.

Поэтому для решения задачи надо организовать цикл проверки чётности значений информационных частей всех элементов исходного списка, начиная с головного (*while*(*beg*)). Если остаток от деления значения информационной части на 2 не равен нулю (*if*(*beg*->*data*%2)), то это значение в качестве информационной части очередного элемента направляется в список с нечётными значениями, в противном случае – в другой список.

После этого переходим к проверке значения информационного поля следующего элемента (*beg* = *beg*->*link*), и так до конца исходного цикла.

Схема алгоритма функции *obrabotka*() приведена на рисунке.

***Схема алгоритма функции obrabotka()***

*obrabotka*(*beg,&beg*0,*&end*0,*&beg*1,*&end*1)

нет

*beg*

да

*beg*->*data % 2*!=0

нет

да

*add\_spis*(*beg->data*,*beg*0,*end*0)

*add\_spis*(*beg->data*,*beg*1,*end*1)

*beg*=*beg*->*link*

*return*

**В основной файл *spis*1.*cpp*** помещаем функцию *main*().

Исполнение программы (функции *main*()) начинается с объявления переменных. Это переменная *ch* типа *telem* и 6 указателей на первый и последний элементы каждого из трёх списков (одного исходного и двух результирующих).

Инициализируем исходный и два результирующих списка. Для этого в функцию *main*() помещаем три вызова функции *init\_spis*(), параметрами которой являются адреса указателей на первый и последний элементы инициализируемого списка (&*l\_beg*, &*l\_end*).

Приступаем к созданию исходного линейного списка. Поскольку линейный список – это динамическая структура, при создании пользователю выводится подсказка с указанием признака окончания ввода. У элементов списка целочисленная информационная часть, поэтому для примера оговариваем, что признаком окончания ввода значений является значение 0.

Пользователь вводит значения одной строкой через пробел и нажимает *Enter*. Считываем первое введённое значение, и если оно отлично от 0, организуем цикл *while*(*ch*). В тело этого цикла помещаем вызов функции добавления нового элемента в список *add*\_*spis*(), передавая ей в качестве параметров значение очередного считанного значения *ch* и адреса указателей на первый и последний элементы списка (&*l\_beg*, &*l\_end*).

После выхода из цикла проверяем факт создания исходного списка, сравнивая значение указателя на первый элемент списка со значением *NULL*. Если список не создан, выполнение программы заканчивается (*return* 0).

Если список создан, выводим значения его элементов, вызывая функцию *view\_spis*(), параметром которой является значение указателя на первый элемент созданного исходного списка *l\_beg*.

Проверка создания списка выполнена. Приступаем к обработке созданного списка. Вызываем функцию *obrabotka*(), передавая ей в качестве параметров значение указателя на первый элемент исходного списка *l\_beg* и указатели на адреса головного и концевого элементов результирующих списков.

После обработки списка и формирования списков с чётными и нечётными значениями информационных частей по очереди выводим значения элементов этих списков, вызывая функцию *view\_spis*(), параметром которой является значение указателя на первый элемент сформированного списка *l\_beg*.

Программа завершается. Удаляем все списки из динамической памяти, трижды вызывая функцию *udal\_spis*() и передавая ей в качестве параметров значения адресов указателей на первый и последний элементы списков (&*l\_beg,* &*l\_end*).

Анализируем результаты работы проекта и делаем выводы.

Схема алгоритма функции *main*() приведена на рисунке.

***Схема алгоритма функции main()***

начало

*init\_spis*(&*l\_beg,*&*l*\_*end*)

*init\_spis*(&*l\_beg*0,&*l*\_*end*0)

*init\_spis*(&*l\_beg*1,&*l*\_*end*1)

ввод

*ch*

нет

*ch*!=0

да

*add\_spis*(*ch*,&*l\_beg*,&*l*\_*end*)

ввод

*ch*

нет

*l\_beg*!=0

да

*view\_spis*(*l\_beg*)

*obrabotka*()

*view\_spis*(*l\_beg*1)

*view\_spis*(*l\_beg*0)

*udal\_spis*(&*l\_beg*,&,*l\_end*)

*udal\_spis*(&*l\_beg*1,&,*l\_end*1)

*udal\_spis*(&*l\_beg*0,&,*l\_end*0)

конец

***Текст программ проекта (пример 2)***

***Заголовочный файл modSp.h***

/\*Заголовочный файл moSp1.h

Содержит определения новых типов и объявления функций

для работы со списком\*/

//Определения типов

typedef int telem ; //определение типа информационного поля

struct list //определение типа элемента списка

{telem data ; //информационное поле

list \*link ; //поле адреса

} ;

//Объявления (прототипы) функций

void init\_spis(list\*\*, list\*\*) ;

void add\_spis(telem, list\*\*, list\*\*) ;

void obrabotka(list\*, list\*\*, list\*\*, list\*\*, list\*\*) ;

void view\_spis(list\*) ;

void udal\_spis(list\*\*, list\*\*) ;

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Основной файл spis1.cpp***

/\*Создается линейный список целых чисел.

После этого программа разбивает исходный список

на два списка - из четных и нечетных значений.

Многофайловый проект.

Основной файл spis1.cpp\*/

#include <iostream.h>

#include"moSp1.h"

using namespace std ;

//main\_beg\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//основная программа

int main()

{telem ch ;

list \*l\_beg, \*l\_end, \*l\_beg0, \*l\_end0, \*l\_beg1, \*l\_end1 ;

setlocale(LC\_ALL,”Russian”) ;

//инициализация списков

init\_spis(&l\_beg, &l\_end) ;

init\_spis(&l\_beg0, &l\_end0) ;

init\_spis(&l\_beg1, &l\_end1) ;

cout << "\n Создается линейный список целых чисел." << endl

<< " После этого исходный список разбивается" << endl <<

<< " на два списка – из четных и нечетных значений" ;

//создание исходного списка

cout << "\n Вводите элементы исходного списка" << endl <<

" через пробел одной строкой; в конце - ноль:" << endl ;

cin >> ch ;

while(ch)

{add\_spis(ch, &l\_beg, &l\_end) ;

cin >> ch ;

}

if(l\_beg) //если список создан (не пуст)

{cout << " Исходный список:" << endl ;

//выводим исходный список на экран

view\_spis(l\_beg) ;

//обрабатываем созданный список и получаем два результирующие

obrabotka(l\_beg, &l\_beg0, &l\_end0, &l\_beg1, &l\_end1) ;

cout << "\n Результирующий список с нечетными значениями:" << endl ;

view\_spis(l\_beg1) ;

cout << "\n Результирующий список с четными значениями:" << endl ;

view\_spis(l\_beg0) ;

//удаляем списки из динамической памяти

udal\_spis(&l\_beg, &l\_end) ;

udal\_spis(&l\_beg0, &l\_end0) ;

udal\_spis(&l\_beg1, &l\_end1) ;

}

else cout << " Исходный список не создан" ;

return 0 ;

}

//main\_end\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Исходный файл modSp.cpp***

/\*Исходный файл moSp1.cpp

Реализация набора функций для работы со списком\*/

#include<iostream.h>

#include"moSp1\_20.h"

using namespace std ;

setlocale(LC\_ALL,”Russian”) ;

//add\_spis\_beg\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//Добавление нового элемента в конец списка

void add\_spis(int ch, list \*\*beg, list \*\*end)

{list \*nov = new list ; //Выделяем память под элемент списка.

nov->data = ch ; //Заполняем информационную часть.

nov->link = 0 ; //В ссылочную - NULL.

if(\*beg) //Если список не пуст,

(\*end)->link = nov ; //добавляем в конец списка.

else \*beg = nov ; //Если список был пуст.

\*end = nov ; //Новый элемент - последний.

return ;

}

//add\_spis\_end\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//obrabotka\_beg\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//построение результирующих списков

void obrabotka(list\*beg,list\*\*beg0,list\*\*end0,list\*\*beg1,list\*\*end1)

{while(beg)

{if(beg->data % 2)

add\_spis(beg->data, beg1, end1) ;

else add\_spis(beg->data, beg0, end0) ;

beg = beg->link ;

}

return ;

}

//obrabotka\_end\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//view\_spis\_beg\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//просмотр списка

void view\_spis(list \*beg)

{while(beg)

{cout << beg->data << ' ' ;

beg = beg->link ;

}

return ;

}

//view\_spis\_end\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//init\_spis\_beg\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//инициализация списка

void init\_spis(list \*\*beg, list \*\*end)

{\*beg = \*end = 0 ; //исходный список пуст

return ;

}

//init\_spis\_end\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//udal\_spis\_beg\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//инициализация списка

void udal\_spis(list \*\*beg, list \*\*end)

{list \*tec ;

while(\*beg)

{tec = \*beg ;

\*beg = (\*beg)->link ;

delete tec ;

}

\*end = 0 ;

//список пуст

return ;

}

//udal\_spis\_end\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Скриншот результатов выполнения программы (пример 2)***

