**Большое домашнее задание (БДЗ)**

## Создание списка объектов

**Цель работы:**  Изучить методы работы с однонаправленным списком. Вспомнить практические навыки работы с файлами. Использовать знания и практические навыки, полученные в течение семестра для решения задач с использованием списков объектов класса.

## Теоретические сведения

При подготовке используйте информацию из лекций предыдущего семестра (выложено в папке)

Списки.

Список является динамической структурой, состоящей из элементов двух типов – информационных и ссылочных. Ссылочные поля содержат указатели на следующий элемент списка.



Для организации списка необходимо создать структуру следующего вида:

struct list { int info; // информационный элемент (поля данных)

struct list \*next; //указатель следующий элемент

};

В качестве информационного поля может быть одна или несколько переменных любого типа. Типом указателя на следующий элемент является свой собственный тип структуры ( в нашем случае struct list).

Создание списка.

Для создания списка необходимо выполнить следующие действия:

1. Объявляем структуру списка:

struct list { int info;

struct list \*next;

};

1. Вводим новых тип данных - указатель на созданную структуру:

typedef list \*P\_list;

Теперь, когда мы захотим создать указатель на только что созданную структуру( struct list) можно использовать тип P\_list, иными словами следующие две записи эквивалентны:

struct list \*ptr; // ptr – это указатель на тип list

P\_list ptr;

1. Для работы со списком требуется, по крайней мере, два указателя:
   * Указатель на начало списка;
   * Указатель на текущий элемент списка.

Перед началом работы оба указателя необходимо объявить:

P\_list head; // указатель на начало списка

P\_list tek; // указатель на текущий элемент списка

1. Создать первый элемент списка, который часто называют заголовком:

head= new list; // выделить память под первый элемент

head ->dat = 1; // записать данные

tek= head ; // текущий указатель установить на начало

1. Создать все последующие элементы списка. Эти инструкции как правило выполняются многократно ( по числу элементов списка).

tek->next= new list;// выделить память под очередной элемент

tek=tek->next; // текущий указатель установить на только что созданный элемент

tek->dat = 2; // записать данные в информационное поле

1. После того, как формирование списка закончено, записать признак конца в указатель последнего элемента.

tek->next=NULL;

Традиционно нулевое значение указателя отмечает конец списка. Нулевой указатель в заголовке списка является признаком того, что список пуст. В дальнейшем, при обработке списка, нулевое значение указателя служит сигналом к окончанию работы.

Пример 5: Создать список, элементами которого являются целые числа (полученные случайным образом), размер списка вводится с клавиатуры.

#include <iostream>

#include <time.h>

#include <stdlib.h>

using namespace std;

struct list // определение структурного типа list

{ int dat; // данные

struct list \*next; // указатель на следующий эл-т списка

};

typedef list \*P\_list; // определение указателя на тип list

void main()

{ int N,i; // N-количество элементов в списке

P\_list pb; // указатель на начало списка

P\_list pt; // указатель на текущий элемент списка

time\_t t;

srand (time(&t)); // инициировать датчик случ. чисел

cout << " N="; // ввести размер списка (с клавиатуры)

cin >> N;

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Создание списка

// создать первый элемент

pb= new list; // выделить память

pb->dat = rand()%100; // записать данные

pt=pb; // установить текущий указатель на начало

for (i=1;i<N;i++) // в цикле создать список

{pt->next= new list;// очередной элемент (выделить память)

pt=pt->next; //текущий указатель-на вновь созданный элемент

pt->dat = rand()%100; // записать данные

} // создание списка завершено

pt->next=NULL; // записать признак конца списка

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ печать списка в 10 колонок

cout<<endl;

for (pt=pb,i=1; pt!= NULL ;i++)

{cout<< pt->dat; // печать информационного поля списка

pt=pt->next; //перемещение указателя на следующий элемент

if (i%10==0) cout<<endl; // форматирование (в 10 колонок)

else cout<<"\t";

}

cout<< endl;

}

Смотрите еще примеры работы со списками : Дополнительные примеры / списки

**Работа с файлами**

Временные данные, хранящиеся в переменных и массивах необходимо сохранять в долговременной памяти – файлах, расположенных на внешних носителях Файлы имеют имя (состоящее из названия и возможного расширения, отделяемого точкой) и хранятся на дисках и других внешних запоминающих устройствах.

Язык СИ рассматривает любой файл как последовательность байтов. Каждый файл оканчивается или маркером конца файла (EOF) или особым байтом, определенным в программе пользователя.

байт 0 байт 1 байт 2 … EOF

Для работы с файлом необходимо выполнить некоторые, одинаковые для всех случаев действия:

**Действие 1.** **Объявить указатель на структуру FILE**

Работа с файлом начинается с объявления указателя на структуру FILE (определенную в <stdio.h>).

**Пример** такого объявления:

FILE \*file1;

где file1 – имя указателя.

Если необходимо использовать несколько файлов, то потребуются указатели для каждого из них.

открыть файл

**Действие 2.** **Открыть файл**

Для работы с файлом необходимо установить связь между программой и физическим файлом (открыть файл), присвоив конкретное значение указателю. Для этого служит функция fopen(…,…), которой передаются два аргумента: имя файла и режим доступа к файлу.

Существуют следующие режимы доступа и соответствующие

им параметры:

* r – открывает уже существующий файл для чтения;
* w – создает и открывает новый файл для записи;
* a – открывает для добавления (дозаписи) информации в конец файла.

Также многие компиляторы СИ позволяют открывать файлы

одновременно и для чтения и для записи. Для этого в аргументе

функции fopen() указываются следующие параметры:

* r+ – открывает уже существующий файл для чтения и для записи;
* w+ – создает и открывает новый файл для чтения и записи;
* a+ – открывает для чтения и записи (в случае существования файла –

для дозаписи в его конец).

**Пример:**

file1=fopen("data.dat","w");

указатель file1 связывается с файлом data.dat, который будет открыт или создан для записи.

Выделяют два вида файлов – текстовые и бинарные файлы.

**Текстовый файл** – файл, содержащий текст, разбитый на строки специальными кодами «возврат каретки» и «перевод строки». Если файл открыт в текстовом режиме, то при чтении из файла комбинация символов «возврат каретки – перевод строки» преобразуется в один символ \n – переход к новой строке. При записи в файл осуществляется обратное преобразование.

**Бинарный файл** – файл, из которого байты считываются и выводятся в первоначальном виде без каких-либо преобразований.

Все указанные выше параметры режимов открывают текстовые файлы. Если требуется указать на двоичный файл, то к параметру добавляется буква b. **Например:** rb, или wb, или r+b.

**Действие 3.** Работа с файлом

Функции работы с файлами:

* fputс(переменная типа char, указатель на файл) – посимвольная запись данных в файл.
* fgetc(указатель на файл) – посимвольное чтение из файла.
* fputs(переменная типа string, указатель на файл) – построчная запись данных в файл. Записывает в файл строку, но в конце не добавляет символ окончания строки.
* fgets(переменная типа string, длина, указатель на файл) – построчное чтение данных из файла. Читает строку целиком до символа новой строки, если ее длина не превышает значения параметра «длина» минус один символ. Параметр «длина» является целым числом или целочисленной переменной, указывающей максимально возможное количество символов в строке.
* fprintf(указатель на файл, строка формата, список переменных) – форматированная запись символов, строк или чисел в файл.
* fscanf(указатель на файл, строка формата, список переменных) – форматированный ввод символов строк или чисел из файла.
* fwrite(указатель на буфер хранения данных, размер элемента, количество элементов, указатель на файл) – запись заданного количества блоков данных определѐнной длины из буфера в файл.
* fread(указатель на буфер размещения данных, размер элемента, количество элементов, указатель на файл) – чтение блоков данных заданного размера в указанном количестве из файла в буфер.
* feof(указатель на файл) – функция определяет, достигнут ли конец файла. Если текущая позиция является концом файла (EOF), то функция возвращает ненулевое значение, в противном случае возвращается 0.
* remove(имя файла) – удаляет файл. Функция remove()ввозвращает 0, если файл успешно удален.
* rename(старое имя, новое имя) – переименовывает файл или директорию, указанную в параметре «старое имя», и присваивает имя, указанное в параметре «новое имя». Также может применяться для перемещения файла.
* fseek(указатель на файл, количество байт, начало отсчета) – устанавливает указатель текущей позиции в файле. Количество байт отсчитывается от значения параметра «начало отсчета», оно определяет новое значение указателя текущей позиции, а начало отсчѐта – это один из следующих макросов: начало файла (SEEK\_SET), текущая позиция (SEEK\_CUR), конец файла (SEEK\_END). Обычно данная функция применяется только для бинарных файлов.

В языке Си определены два стандартных файла ввода-вывода, к

которым можно обращаться с помощью следующих указателей:

* stdin – указатель на стандартный поток ввода. По умолчанию ему ставится в соответствие клавиатура.
* stdout – указатель на стандартный поток вывода. По умолчанию ему соответствует экран дисплея.

**Действие 4. Закрыть файл**

После окончания работы с файлом, его требуется закрыть, то есть прервать связь между файлом и программой. Для этого служит функция fclose(…), которой необходимо передать указатель на закрываемый файл.

**Пример:**

fclose (file1); /\* закрывает файл data.dat, связанный с

указателем file1 \*/.

**Пример:** **Работа с текстовыми файлами.**

Открыть существующий текстовый файл first.txt, посимвольно считать его содержимое и записать это содержимое в новый текстовый файл second.txt.

#include <stdio.h>

#include <stdlib.h>

void main( )

{

FILE \*f1,\*f2;

int ch;

if ((f1 = fopen("first.txt","r")) == NULL)

{

fprintf(stdout, "Error opening file c\_rec");

exit(1); /\* выход из программы в случае ошибки открытия файла first.txt\*/

}

if ((f2 = fopen("second.txt","w")) == NULL)

{

fprintf(stdout, "Error opening file b\_rec");

exit(1);

}

while(feof(f1)==0)

{

ch = fgetc(f1); /\* посимвольное чтение из first.txt и запись в second.txt, пока не

встретиться конец файла\*/

fputc(ch,f2);

fputc(ch,stdout);

}

fclose(f1);

fclose(f2);

}

**Пример:** **Работа с бинарными файлами.**

В исходном бинарном файле input.dat записаны 30 целых чисел типа int, значения которых лежат в диапазоне от -20 до 20. Необходимо считать данные из файла input.dat и записать все положительные числа в бинарный файл output.dat.

#include <stdio.h>

#define N 30

void main()

{

int i, n, A[N];

FILE \*fp;

fp = fopen("input.dat","rb");

n = fread(A, sizeof(int), N, fp); /\*чтение блока данных в массив\*/

fclose(fp);

fp = fopen("output.dat","wb");

for(i=0;i<N;i++)

{

if(A[i]>0)

fwrite(&A[i],sizeof(int),1,fp); /\*запись данных по одному элементу из массива\*/

}

fclose(fp);

}

**Общие требования к заданиям**

1. При оформлении ввода-вывода данных информация на экране должна быть отформатирована:

* на экран выводится тема задания (кратко);
* ввод данных и результат вычислений выводить с комментариями;
* выделять области ввода и вывода информации с помощью строк-разделителей.

1. Во всех заданиях необходимо организовать линейный связанный список из класса, указанного в задании.
2. Все данных класса должны быть защищенными.
3. Со списком необходимо выполнить следующие действия:

* Чтение информации из файла и размещение её в памяти (создание списка);
* Просмотр списка (вывод информации на экран в отформатированном виде);
* Включение нового элемента в заданное место списка (место включения указано в задании). Данные для нового элемента вводятся с клавиатуры.
* Удаление заданного элемента (место удаления указано в задании)

1. Выполнение операций над списком должно осуществляться в интерактивном режиме, то есть необходимо создать меню команд и возможность многократного их выполнения.
2. После выхода из меню, измененный список записывается в тот же файл, который был открыт ранее.
3. Действия над списками реализуются в виде методов класса дружественных функций, а также с помощью перегрузки операторов. Варианты реализации указаны в задании.

## Варианты заданий

|  |  |
| --- | --- |
| Вариант |  |
| 1,16 | Организовать линейный связанный список из объектов класса Человек с полями:   * имя (char\* или string) * фамилия (char\* или string) * возраст (int)   Дополнение к интерфейсу класса (общие требования смотри выше):   * Включить элемент в начало списка (дружественная функция) * Удалить элемент, задается номер удаляемого элемента (перегрузить оператор -) |
| 2,17 | Организовать линейный связанный список из объектов класса Работник с полями:   * номер бригады (int) * фамилия (char\* или string) * разряд (int)   Дополнение к интерфейсу класса (общие требования смотри выше):   * Включить элемент, место указывается номером, который должен иметь элемент после включения.   (перегрузить оператор +)   * Удалить первый элемент списка (метод класса) |
| 3,18 | Организовать линейный связанный список из объектов класса Студент с полями:   * номер группы (int) * фамилия (char\* или string) * размер стипендии (double)   Дополнение к интерфейсу класса (общие требования смотри выше):   * Включить элемент (перегрузить оператор +) * Удалить последний элемент списка (дружественная функция) |
| 4,19 | Организовать линейный связанный список из объектов класса Фирма   * название фирмы (char\* или string) * адрес (char\* или string) * количество сотрудников (int)   Дополнение к интерфейсу класса (общие требования смотри выше):   * Включить элемент в конец списка (перегрузить оператор +) * Удалить второй элемент списка (метод класса) |

|  |  |
| --- | --- |
| 5,20 | Организовать линейный связанный список из объектов класса Книга с полями:   * Название (char\* или string) * Автор (char\* или string) * Тираж (int)   Дополнение к интерфейсу класса (общие требования смотри выше):   * Включить элемент в начало списка (дружественная функция) * Удалить первый элемент списка (перегрузить оператор -) |
| 6,21 | Организовать линейный связанный список из объектов класса Преподаватель   * Фамилия (char\* или string) * Должность (char\* или string) * Стаж (int)   Дополнение к интерфейсу класса (общие требования смотри выше):   * Включить элемент в конец списка (перегрузить оператор +) * Удалить элемент, задается номер удаляемого элемента   (метод класса) |
| 7,22 | Организовать линейный связанный список из объектов класса Файл с полями:   * название файла (char\* или string) * тип данных (char\* или string) * количество байт (int)   Дополнение к интерфейсу класса  (общие требования смотри выше):   * Включить элемент, место указывается номером, который должен иметь элемент после включения (метод класса) * Удалить последний элемент списка (перегрузить оператор -) |
| 8,23 | Организовать линейный связанный список из объектов класса Транспортное средство с полями:   * Название транспортного средства (char\* или string) * номер транспортного средства (int) * номер маршрута (int)   Дополнение к интерфейсу класса (общие требования смотри выше):   * Включить элемент в начало списка (дружественная функция) * Удалить элемент, задается номер удаляемого элемента (перегрузить оператор -) |

|  |  |
| --- | --- |
| 9,24 | Организовать линейный связанный список из объектов класса Статья с полями:   * Автор (char\* или string) * Заглавие (char\* или string) * Журнал (char\* или string) * номер журнала (int) * год выпуска (int)   Дополнение к интерфейсу класса (общие требования смотри выше):   * Включить элемент в конец списка (перегрузить оператор +) * Удалить первый элемент списка (дружественная функция) |
| 10,25 | Организовать линейный связанный список из объектов класса Компьютер с полями:   * Марка (char\* или string) * Объем памяти (int) * Название (char\* или string)   Дополнение к интерфейсу класса (общие требования смотри выше):   * Включить элемент в начало списка (перегрузить оператор +) * Удалить последний элемент списка (метод класса) |
| 11,26 | Организовать линейный связанный список из объектов класса Животное   * Род (char\* или string) * Название (char\* или string) * место обитания (char\* или string)   Дополнение к интерфейсу класса (общие требования смотри выше):   * Включить элемент и разместить его на 4 месте списка (перегрузить оператор +) * Удалить элемент, задается номер удаляемого элемента (дружественная функция) |
| 12,27 | Организовать линейный связанный список из объектов класса Оружие с полями:   * Калибр (int) * Тип (char\* или string) * фирма изготовитель (char\* или string)   Дополнение к интерфейсу класса (общие требования смотри выше):   * Включить элемент, место указывается номером, который должен иметь элемент после включения (метод класса) * Удалить элемент, задается номер удаляемого элемента (перегрузить оператор -) |

|  |  |
| --- | --- |
| 13,28 | Организовать линейный связанный список из объектов класса Наука с полями:   * предмет исследования (char\* или string) * методы исследования (char\* или string)   Дополнение к интерфейсу класса (общие требования смотри выше):   * Включить элемент в конец списка (перегрузить оператор +) * Удалить второй элемент списка (дружественная функция) |
| 14,29 | Организовать линейный связанный список из объектов класса Авиакомпания с полями:   * название авиакомпании (char\* или string) * капитал (int)   Дополнение к интерфейсу класса (общие требования смотри выше):   * Включить элемент и разместить его на 2 месте списка   (метод класса)   * Удалить элемент, задается номер удаляемого элемента (перегрузить оператор -) |
| 15,30 | Организовать линейный связанный список из объектов класса Программист с полями:   * Фамилия (char\* или string) * Отдел (int) * Язык программирования (char\* или string)   Дополнение к интерфейсу класса (общие требования смотри выше):   * Включить элемент в начало списка (перегрузить оператор +) * Удалить последний элемент списка (метод класса) |