Работа с файлами (язык Си)

Тема: Входящие и исходящие потоки

Что такое поток?

Каждый класс, поддерживающий потоковый ввод-вывод, наследует классы **std::istream** (ввод), **std::ostream** (вывод) или **std::iostream** (ввод и вывод). Именно они обеспечивают возможность использования перегруженных операторов '<<' и '>>', форматирования вывода, преобразование чисел в строки и наоборот и т.д.

Библиотека iostream определяет три стандартных потока:

- cin стандартный входной поток (stdin в C)
- cout стандартный выходной поток (stdout в C)
- cerr стандартный поток вывода сообщений об ошибках (stderr в C)

Для их использования необходимо прописать строку:

```
using namespace std;
```

Как с этим жить?

Для выполнения операций ввода-вывода переопределены две операции поразрядного сдвига:
>> получить из входного потока
<< поместить в выходной поток

Вывод информации
cout << значение;

Здесь значение преобразуется в последовательность символов и выводится в выходной поток:

cout << n;

Как с этим жить?

Особого внимания заслуживает ввод символьных строк. По умолчанию потоковый ввод cin вводит строку до пробела, символа табуляции или перевода строки.

```
#include <iostream>
using namespace std;
int main()
{
  char s[80];
  cin >> s;          // Поток ввода
  cout << s << endl;  // Поток вывода
 }
```

Как с этим жить?

Для ввода текста до символа перевода строки используется манипулятор потока getline():

```
#include <iostream>
using namespace std;
int main()
{
  char s[80];
  cin.getline(s, 80);   //Потоком читаем строку
  cout << s << endl;
}
```

Манипуляторы потока

Функцию - манипулятор потока можно включать в операции помещения в поток и извлечения из потока (<<, >>).
В C++ имеется ряд манипуляторов. Рассмотрим основные:

Манипулятор	Описание
`endl`	Помещение в выходной поток символа конца строки `'\n'`
`dec`	Установка основания 10-ой системы счисления
`oct`	Установка основания 8-ой системы счисления
`hex`	Установка основания 16-ой системы счисления
`setbase`	Вывод базовой системы счисления
`width(ширина)`	Устанавливает ширину поля вывода
`fill('символ')`	Заполняет пустые знакоместа значением символа
`precision(точность)`	Устанавливает количество значащих цифр в числе (или после запятой) в зависимости от использования `fixed`
`fixed`	Показывает, что установленная точность относится к количеству знаков после запятой
`showpos`	Показывает знак + для положительных чисел
`scientific`	Выводит число в экспоненциальной форме
`get()`	Ожидает ввода символа
`getline(указатель, количество)`	Ожидает ввода строки символов. Максимальное количество символов ограничено полем количество

Небольшая проблема - формат

```
double a = -112.234;
double b = 4.3981;
int c = 18;
cout << endl << "double number:" << endl;
cout << "width(10)" << endl;
cout.width(10);
cout << a << endl << b << endl;
cout << "fill('0')" << endl;
cout.fill('0');
cout.width(10);
cout << a << endl << b << endl;
cout.precision(5);
cout << "precision(5)" << endl << a << endl << b << endl;
cout << "fixed" << endl << fixed << a << endl << b << endl;
cout << "showpos" << endl << showpos << a << endl << b << endl;
cout << "scientific" << endl << scientific << a << endl << b << endl;
cout << endl << "int number:" << endl;
cout << showbase << hex << c << " " << showbase << oct << c << " ";
cout << showbase << dec << c << endl;
cin.get();
```

Файловый поток

Для программиста открытый файл представляется как последовательность считываемых или записываемых данных. При открытии файла с ним связывается ***поток ввода-вывода***. Выводимая информация записывается в поток, вводимая информация считывается из потока.

`<ifstream>` — файловый ввод ;
`<ofstream>` — файловый вывод.

```
#include <fstream>
using namespace std;
int main()
{
  ofstream fout;            //Создаем файловый поток
  fout.open("file.txt");    //Открываем файловый поток
  fout << "Привет, мир!";     //Пишем строку в файл
  fout.close();         //Закрываем поток
}
```

Режимы доступа

Режимы открытия файлов устанавливают характер использования файлов. Для установки режима в классе `ios` предусмотрены константы, которые определяют режим открытия файлов.

Константа	Описание
`ios::in`	открыть файл для чтения
`ios::out`	открыть файл для записи
`ios::ate`	при открытии переместить указатель в конец файла
`ios::app`	открыть файл для записи в конец файла
`ios::trunc`	удалить содержимое файла, если он существует
`ios::binary`	открытие файла в двоичном режиме

Как с этим жить?

Режимы открытия файлов можно устанавливать непосредственно при создании объекта или при вызове метода `open()`.

```
ofstream fout("file.txt", ios::app);
fout.open("file.txt", ios::app);
```

Режимы открытия файлов можно комбинировать с помощью поразрядной логической операции **ИЛИ** |, например:

`ios::out | ios::in` - открытие файла для записи и чтения.

Произвольный доступ к файлу

Система ввода-вывода C++ позволяет осуществлять произвольный доступ с использованием методов `seekg()` и `seekp()`.

- `ifstream &seekg(Смещение, Позиция);`
- `ofstream &seekp(Смещение, Позиция);`

Смещение определяет область значений в пределах файла (`long int`).

Система ввода-вывода C++ обрабатывает два указателя, ассоциированные с каждым файлом:

- `get pointer` **`g`** - определяет, где именно в файле будет производиться следующая операция ввода;
- `put pointer` **`p`** - определяет, где именно в файле будет производиться следующая операция вывода.

Произвольный доступ к файлу

```
char s[80];
  fstream inOut;
  inOut.open("file.txt", ios::out);
  inOut << "строчка текста" << endl;
  inOut.seekp(8, ios::beg);
  inOut << "еще строчка текста";
  inOut.close();
  inOut.open("file.txt", ios::in);
  inOut.seekg(-6, ios::end);
  inOut >> s;
  inOut.close();
  cout << s;
  cin.get();
```

Задача

Нужно заполнять таблицу

ФИО	Дата рождения	Хобби

Алгоритм решения задачи следующий:

1. формируем очередную строку для вывода
2. открываем файл для чтения, считываем из него данные и сохраняем их в массив строк
3. закрываем файл
4. открываем файл для записи
5. выводим "шапку" таблицы
6. выводим новую строку
7. выводим все сохраненные строки обратно в файл, начиная со строки после шапки

Целочисленные алгоритмы (язык Си)

Тема: Длинные числа

Что такое длинные числа?

Задача. Вычислить (точно)

$$100! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot 99 \cdot 100$$

Проблема:

это число содержит более 100 цифр…

Сколько нулей в конце этого числа?

Какая последняя ненулевая цифра?

Решение:

хранить цифры в виде массива, по группам (например, 6 цифр в ячейке).

Сколько ячеек нужно?

$$100! < \underbrace{100^{100}}_{\text{201 цифра}}$$

$201/6 \approx 34$ ячейки

Хранение длинных чисел

$$1234\ 568901\ 734567 =$$
$$= 1234 \cdot 1000000^2 + 568901 \cdot 1000000^1 + 734567 \cdot 1000000^0$$

Хранить число по группам из 6 цифр – это значит представить его в системе счисления с основанием **d** = 1000000.

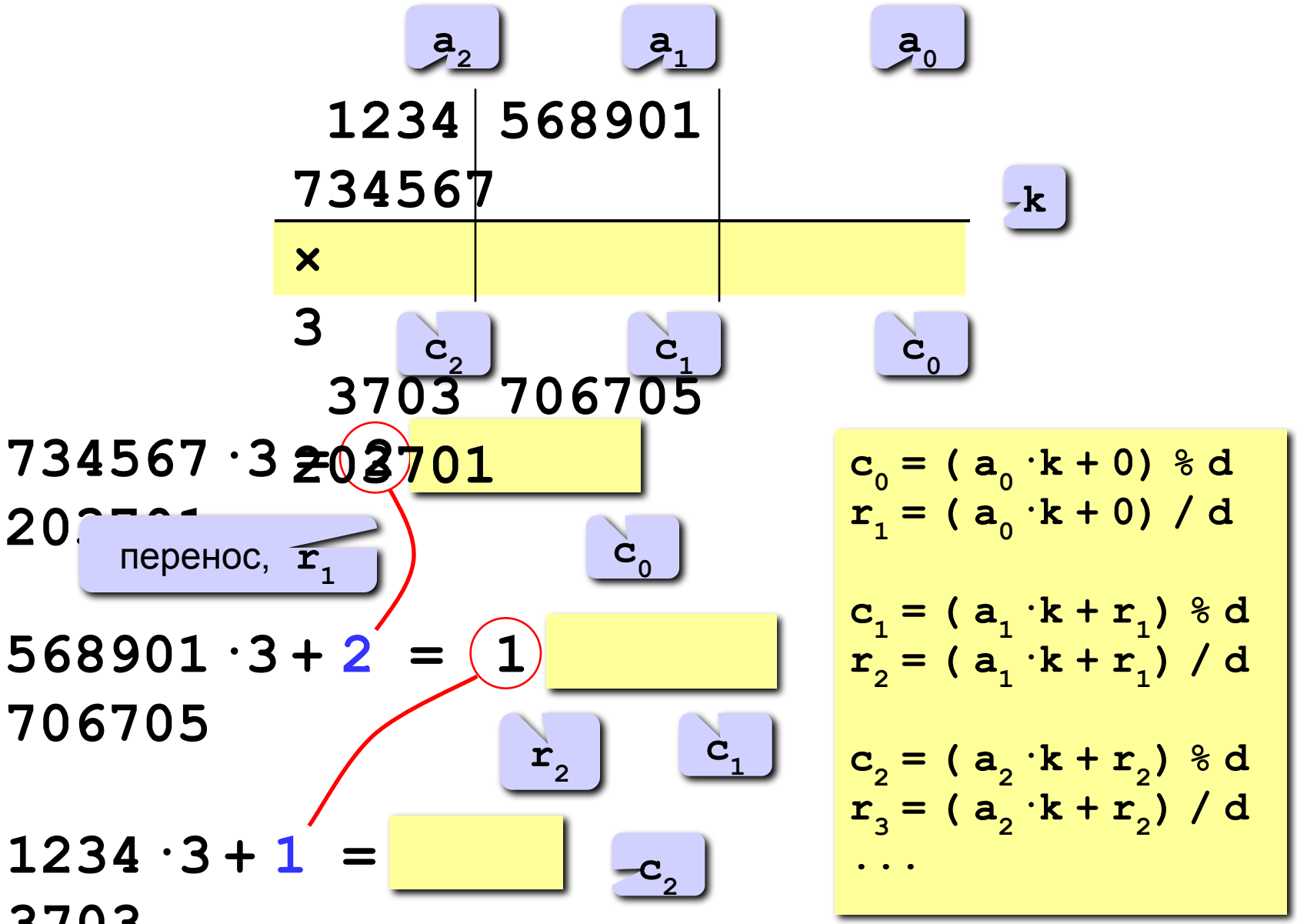
Алгоритм:

```
{A} = 1;
for ( k = 2; k <= 100; k ++ )
   { A} = {A} * k;
... // вывести { A}
```

{A} – длинное число, хранящееся как массив

умножение длинного числа на «короткое»

Умножение длинного числа на короткое



Вычисление 100!

```
const int d = 1000000;    // основание системы
int  A[40] = {1},         // A[0]=1, остальные A[i]=0
     s, r;                // произведение, остаток
int i, k, len = 1;        // len – длина числа

for ( k = 2; k <= 100; k ++ ) {
  i = 0;
  r = 0;
  while ( i < len  ||  r > 0 ) {
    s = A[i]*k + r;
    A[i] = s % d;      // остается в этом разряде
    r = s / d;         // перенос
    i ++;
    }
  len = i;      // новая длина числа
  }
```

пока не кончились цифры числа **{A}** или есть перенос

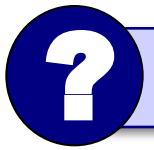
Где результат?

Можно ли брать другое d?

Как вывести длинное число?

«Первая мысль»:

```
for ( i = len-1; i >= 0; i -- )
  printf ( "%d", A[i] );
```



Что плохо?

Проблема:

как не потерять первые нули при выводе чисел, длина которых менее 6 знаков?

123 → 000123

Решение:

1) составить свою процедуру;
2) использовать формат **"%.6d"**!

```
for ( i = len-1; i >= 0; i -- )
  if ( i == len-1 ) printf ( "%ld", A[i] );
  else              printf ( "%.6d", A[i] );
```

Задания

«4»: Составить программу для вычисления

$$99!! = 1 \cdot 3 \cdot \ldots \cdot 97 \cdot 99$$

«5»: То же самое, но написать свою процедуру для вывода (не использовать формат `"%.6d"`).

“6": Написать программу для умножения двух длинных чисел (ввод из файла).

“7": Написать программу для извлечения квадратного корня из длинного числа (ввод из файла).

ООП
(язык Си)

Тема: Структуры и классы

Что такое структуры?

Структура в языке C++ представляет собой производный тип данных, который представляет какую-то определенную сущность, также как и класс. Нередко структуры применительно к C++ также называют классами. И в реальности различия между ними не такие большие.

Для определения структуры применяется ключевое слово **struct**, а сам формат определения выглядит следующим образом:

```
struct person
{
int age;
string name;
}
```

Что с ними делать?

```
struct person
{
int age;
string name;
}
int main()
{
person man;  // Объявим объект структуры
man.name = "Vasia"; // Присвоим объекту имя
man.age = 25; // Присвоим объекту возраст
cout<<"Name: "<< man.name<<" Age: "<< man.age;
}
```

Есть ли что-то похожее? Да, есть!

```
class Person
{
public:         //Методы, к которым можно обращаться извне
    Person(string n, int a)
    {
     name = n; age = a;
    }
    string getName()
    {
        return name;
    }
    int getAge()
    {
        return age;
    }
private:        //Внутренние данные, необходимые для работы класса
    string name;
    int age;
 };
```

Классы, как быть и куда идти?

```
struct user
{
public:                  //Методы, к которым можно обращаться извне
    user(string n, int a)
    {
        name = n; age = a;
    }
    string getName()
    {
        return name;
    }
    int getAge()
    {
        return age;
    }
private:                 //Внутренние данные, необходимые для работы класса
    string name;
    int age;
};
 int main()
{
    user tom("Tom", 22); // Объявим объект структуры
    cout << "Name: " << tom.getName() << "\tAge: " << tom.getAge() ;
    }
```

Заголовочные файлы

Для удобства описания классов вынося в отдельные файлы с расширением .h от слова header, которые таки и называют «хидеры» или «заголовочные файлы».

Создадим программу, которая будет заниматься учетом успеваемости студентов в группе.
Сформируем заголовочный файл **students.h**, в котором будет находиться класс Students.
Затем обратимся к нему из главной функции основной программы.

(примеры вынесены в отдельные файлы **students.h** и **main.cpp**)

Задания

«4»: Создайте структуру с именем train, содержащую поля: название пункта назначения, номер поезда, время отправления.

1. Ввести данные из файла (создаем заранее) в массив из пяти элементов типа train, упорядочить элементы по номерам поездов.
2. Добавить возможность вывода информации о поезде, номер которого введен пользователем.
3. Добавить возможность сортировки массива по пункту назначения, причем поезда с одинаковыми пунктами назначения должны быть упорядочены по времени отправления.

Задания

«ДЗ»: Создайте **класс** с именем train, содержащий поля: название пункта назначения, номер поезда, время отправления.

1. Ввести данные из файла (создаем заранее) в массив из пяти элементов типа train, упорядочить элементы по номерам поездов.
2. Добавить возможность вывода информации о поезде (метод класса).
3. Добавить возможность сортировки массива по пункту назначения, причем поезда с одинаковыми пунктами назначения должны быть упорядочены по времени отправления.

Работа с памятью (язык Си)

Тема: Динамическое выделение памяти

Основные положения

Возможно, кто-то пробовал делать так:

```
int n = 10;
int arr[n];
```

И, ясное дело, из этого ничего не получилось =)

Дело в том, что количество выделяемых в памяти ячеек, согласно синтаксису C++ должно быть обозначено константой, причем целочисленной. Это сделано для того, чтобы избежать случайной потери данных в процессе работы программы.

Динамическое выделение памяти (указатели)

```
#include <iostream>
using namespace std;
int main()
{
    int *a = new int; // Объявление указателя для переменной типа int
    int *b = new int(5); // Инициализация указателя
    *a = 10;
    *b = *a + *b;
    cout << "b is " << *b << endl;
    delete b;
    delete a;
}
```

Выделение памяти осуществляется с помощью оператора new и имеет вид: тип_данных *имя_указателя = new тип_данных;, например int *a = new int;. После удачного выполнения такой операции, в оперативной памяти компьютера происходит выделение диапазона ячеек, необходимого для хранения переменной типа int.

Для того, чтобы освободить память, выделенную оператором **new**, используется оператор delete.

Динамическое выделение памяти (указатели)

Поскольку массивы это тоже переменные, то правила выделения памяти под них остаются прежними, нужно только учесть. Что элементов, как правило, несколько.

```
#include <iostream>
using namespace std;
int main()
{
    int num; // размер массива
    cout << "Enter integer value: ";
    cin >> num; // получение от пользователя размера массива

    int *p_darr = new int[num]; // Выделение памяти для массива
    for (int i = 0; i < num; i++) {
        // Заполнение массива и вывод значений его элементов
        p_darr[i] = i;
        cout << "Value of " << i << " element is " << p_darr[i] << endl;
    }
    delete [ ] p_darr; // очистка памяти
}
```

Задания

«4»: Найти сумму и среднее арифметическое целочисленных элементов динамического массива. Данные поступают в массив из файла с числами, количество которых заранее неизвестно.

«5»: Заданы два одномерных целочисленных массива A и B, состоящие из N и M элементов соответственно (где $0 \le i < N$; $0 \le i < M$).

Сформировать массив C, элементами которого являются натуральные числа – индексы элементов массива A, значения которых равны элементу B[i] (где $0 \le i < M$), применяя для поиска индекса метод двоичного поиска. При поиске исключать элементы кратные 3.

Возвращаемся к классам

При создании объекта, лучше не копировать память для него, а выделять ее в «общей куче» с помощью указателя. И освобождать ее после того, как мы закончили работу с объектом. Реализуем это в нашей программе, немного изменив содержимое файла **main.cpp**.

(примеры вынесены в отдельные файлы **students.h** и **main2.cpp**)

NOTA BENE

При создании статического объекта, для доступа к его методам и свойствам, используют операция прямого обращения — «**.**» (символ точки). Если же память для объекта выделяется посредством указателя, то для доступа к его методам и свойствам используется оператор косвенного обращения — «**->**».

Конструктор и деструктор класса

Конструктор класса — это специальная функция, которая автоматически вызывается сразу после создания объекта этого класса. Он не имеет типа возвращаемого значения и должен называться также, как класс, в котором он находится. По умолчанию, заполним двойками массив с промежуточными оценками студента.

```
class Students {
    public:
        // Конструктор класса Students
        Students(int default_score)
        {
            for (int i = 0; i < 5; ++i) {
                scores[i] = default_score;
            }
        }
    private:
        int scores[5];
};
int main()
{
    Students *student = new Students(2);
}
```

Конструктор и деструктор класса

Деструктор класса вызывается при уничтожении объекта. Имя деструктора аналогично имени конструктора, только в начале ставится знак тильды **~**. Деструктор не имеет входных параметров.

```
#include <iostream>
class Students {
   public:
      // Деструктор
      ~Students()
      {
         std::cout << "Memory has been cleaned. Good bye." << std::endl;
      }
};

int main()
{
   Students *student = new Students;
   // Уничтожение объекта
   delete student;
}
```

Задания

«!!!»:Необходимо создать класс — зоомагазин. В классе должны быть следующие поля: животное (напр. волк, пингвин, собака), пол, имя , цена, количество. Включить в состав класса необходимый минимум методов, обеспечивающий полноценное функционирование объектов указанного класса:

1. Конструкторы;
2. Деструктор;
3. Предоставить возможность вводить данные с клавиатуры или из файла (по выбору пользователя).
4. Организовать программу учета выручки с продаж.

Динамическое выделение памяти (относительные указатели)

```
int main()
{
    int i, j, N, M;
    int** matrix;
    cin >> M >> N;
    matrix = new int*[M];
    for ( i = 0; i < M; i++)
        matrix[i] = new int[N];
    for ( i = 0; i < M; i++)
        for ( j = 0; j < N; j++)
        {
            cout << «Element of array " << "[" << i << "][" << j << "]  ";
            cin >> matrix[i][j];
        }
        cout << endl;
        for ( i = 0; i < M; i++)
        {
            for ( j = 0; j < N; j++)
       cout << matrix[i][j];
        cout << endl;
        }}
```

Динамическое выделение памяти

(Функция malloc)

Прототип функции malloc:
void * malloc(size_t «размер выделенного блока в байтах»);

Функция malloc выделяет блок памяти, размером sizemem байт, и возвращает указатель на начало блока. Содержание выделенного блока памяти не инициализируется, оно остается с неопределенными значениями.

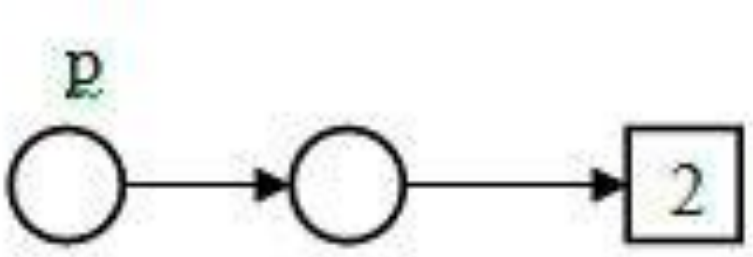
Динамическое выделение памяти

(Функция malloc)

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
  int l;
cout << "String length: ";
cin >> l;
  char * buf = (char*) malloc(l + 1);
  if (buf==0) exit (1);
  for (int ix = 0; ix < l; ix++)
    buf[ix] ='a' + rand() % 26;
  buf[l]='\0';
cout << "Random string: " << buf << "n";
  free(buf);
}
```

Задания

«4»: Дан указатель: double **p = 0;
Выполните следующие задания (решения можно оформлять внутри функции main): * создайте конструкцию, изображенную на рисунке; * выведите число, указанное в квадратике, на экран; * после этого удалите все динамические объекты.



Задания

«5»:Объявите указатель на массив и предложите пользователю выбрать его размер. Далее напишите четыре функции:
первая должна выделить память для массива,
вторая – заполнить ячейки данными,
третья – показать данные на экран,
четвертая – освободить занимаемую память.

Программа должна предлагать пользователю продолжать работу (создавать новые динамические массивы) или выйти из программы.

Задания

«!!!»: Объявите указатель на массив типа int и выделите память для 12-ти элементов. Необходимо написать функцию, которая поменяет значения четных и нечетных ячеек массива. Например есть массив из 4-х элементов:

	ячейка 0	ячейка 1	ячейка 2	ячейка 3
Исходные данные массива	1	2	3	4

	ячейка 0	ячейка 1	ячейка 2	ячейка 3
Данные после работы функции	2	1	4	3

Задания

«ДЗ»: Объявить и заполнить двумерный динамический массив случайными числами от 10 до 50. Показать его на экран. Для заполнения и показа на экран написать отдельные функции. (подсказка: функции должны принимать три параметра – указатель на динамический массив, количество строк, количество столбцов). Количество строк и столбцов выбирает пользователь.

Задания

«!!!»:Создайте игру по модели «Морской бой», однако все корабли будут одинаковые по размеру (10 шт), но каждый корабль имеет уровень прочности (задается случайно 50-100 при установке корабля на поле. Игроки ходят по очереди без дублирования ходов (в случае попадания урон вычитается из прочности корабля (рассчитывается случайно, как и прочность)).

- Размеры поля задаются при начале игры (динамический массив)
- Для описания кораблей (их состояния и координат) использовать структуру или класс

Динамическое выделение памяти для структур

Часто появляется проблема создания дополняемых массивов, размер которых не имел бы ограничений и изменялся в зависимости от количества введенных данных по мере их поступления.

Чаще всего данные имеют смешанный характер, т.е. являются структурами.

В таком случае алгоритм решения задачи таков:

1) Формируется структура
2) Создается массив структур из одного элемента
3) Заполняется один элемент массива
4) Затем делается выбор: ввод или выход
5) Если продолжаем ввод, то создаем дополнительный массив такого-же размера, что и старый и копируем в него данные.
6) Исходный массив удаляется и заново создается размерностью на 1 больше.
7) В пп4.

Задания

«4»: Заполнить динамический массив неотрицательными целыми числами. Ввод прекратить при появлении первого отрицательного числа. Количество вводимых чисел заранее неизвестно.

«ДЗ»: Обеспечить возможность удаления элемента массива из задания 4 по выбору пользователя.

Задания

«4»: Заполнить динамический структурный массив данными из файла(Имя, год рождения, возраст). Ввести возможность упорядочивать данные по всем параметрам. Результат записать в другой файл.

«ДЗ»: Обеспечить возможность поиска элементов в массиве по одному из параметров. Выводить все совпадения в файл.

Векторы

Вектор в C++ — это замена стандартному динамическому массиву, память для которого выделяется вручную, с помощью оператора new.

Основные методы класса vector:
pop_back() — удалить последний элемент
clear() — удалить все элементы вектора
empty() — проверить вектор на пустоту

```
#include <iostream>
#include <vector>
using namespace std;
int main(){
    vector<int> A(10);
    vector<float> B;
    int j=0,N;
    cout<<“Insert size of vector”<<endl;
    cin>>N;
        while(j<N)
        {
            B.push_back(j);
            j++;
        }
     vector<int> C(10, 3);
}
```

Задания

«4»: Создать вектор содержащий ФИО людей, размер не определён, заполнение - вручную, добавить возможность удаления элементов.

«5»: Организовать заполнение вектора из файла с любым количеством строк.

Операции над векторами

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
int main()
{
    vector<string> names;
    string buffer = "";
    cout << "Insert names "<<endl<<"to end the insert write empty
string"<<endl;
    do {
        getline(cin, buffer);
        if (buffer.size() > 0) {
            names.push_back(buffer);
        }
    } while (buffer != "");
    cout << "Yours VECTOR:" << endl;
    for (int i = 0; i < names.size(); i++) {
        cout << names[i] << endl;
    }}
```

Документация по классу vector
http://www.cplusplus.com/reference/vector/vector/

Двумерные массивы при помощи векторов

Для создания двумерного массива при помощи векторов необходимо создать вектор из векторов того типа, который требуется для выполнения задачи. Обращение к элементам происходит аналогично двумерным массивам.

```
vector<vector<int>> V2D;
// Заполнение вектора
cout<<V2D[1][4];
```

Задания

«4»: Объявить класс для хранения данных про лампочки (тип (светодиодная, энергосберегающая или лампа накаливания), мощность (Вт), цвет света, стоимость).
Создать вектор из элементов класса, созданного для хранения информации о лампочках.

«5»: Создать метод TEST для определения и вывода информации.
Определить количество ламп с мощностью меньше 40Вт и вывести информацию о них.