Работа с файлами

Открытие и закрытие файла

- Перед началом работы с файлом его надо создать (открыть), а по окончании работы закрыть.
- Перед началом работы с файлом надо создать указатель на структуру данных типа **FILE.**
- Затем необходимо вызвать функцию **fopen(),** которая может создать новый файл для записи в него, либо открыть су ществующий на диске файл для записи или (и) чтения.

Функции для работы с файлами

Функция	Действие функции
fopen()	Открыть файл
fclose()	Закрыть файл
fputc()	Записать символ в файл
fgetc()	Прочитать символ из файла
fseek()	Изменить указатель позиции файла на указанное место
fprintf()	Форматная запись в файл
fscanf()	Форматное чтение из файла
feof()	Возвращает значение TRUE, если достигнут конец файла
ferror()	Возвращает значение FALSE, если обнаружена ошибка

Функции для работы с файлами

Функция	Действие функции
fread()	Читает блок данных из потока
fwrite()	Пишет блок данных в поток
rewind()	Устанавливает указатель позиции файла на начало
remove()	Уничтожает файл

Основы объектно-ориентированного программирования

Повышение степени абстракции программы

- Шаг 1 – Использование функций
- Шаг 2 - Описание собственных типов данных
- Шаг 3 – Объединение в модули описаний типов данных и функций

Цель повышения уровня абстракции – представление структуры программы в виде меньшего количества более крупных блоков и минимизация связи между ними.

Определения

- Объектно-ориентированное программирование (ООП) — подход к программированию, при котором основными концепциями являются понятия объектов и классов.
- Объектно-ориентированное программирование, ООП - это методика, которая концентрирует основное внимание программиста на связях между объектами, а не на деталях их реализации.

Класс

- В классе структуры данных и функции их обработки объединяются.
- Класс является типом данных, определяемым пользователем.
- В классе задаются свойства и поведение какого-либо предмета или процесса в виде полей данных (аналогично структуре) и функций для работы с ними.
- Существенным свойством класса является то, что детали его реализации скрыты от пользователей класса за интерфейсом.
- Интерфейс класса - заголовки его методов.

Терминология

- *Объект* или *экземпляр класса* – конкретная величина типа данных *класс.*
- *Сообщение* – запрос на выполнение действия, содержащий набор необходимых параметров. Объекты взаимодействуют между собой, посылая и получая сообщения.

Основные свойства ООП

- инкапсуляция;
- наследование;
- полиморфизм.

Инкапсуляция

- *Инкапсуляция* – объединение данных с функциями их обработки в сочетании со скрытием ненужной для использования этих данных информации. Инкапсуляция повышает степень абстракции программы.

Инкапсуляция

- Инкапсуляция представляет собой механизм, который связывает вместе код и данные и который хранит их от внешнего воздействия и от неправильного использования.
- Именно инкапсуляция позволяет создавать объект.
- Объект представляет собой логическое целое, включающее в себя данные и код для работы с этими данными. Возможно определить часть кода и данных как собственность объекта, которая недоступна извне. На этом пути объект обеспечивает существенную защиту против случайной модификации или некорректного использования таких частных членов объекта.
- Во всех случаях объект представляет собой переменную, тип которой определяется пользователем.

Наследование

- *Наследование* – возможность создания иерархии классов, когда потомки наследуют все свойства своих предков, могут их изменять и добавлять новые. Свойства при наследовании повторно не описываются, что сокращает объем программы.
- Без использования классификации каждый объект должен был бы определять все свои характеристики явным образом.
- На основе классификации объект нуждается только в определении таких качеств, которые отличают его от других объектов этого класса.
- Благодаря механизму наследования объект может характеризоваться в рамках классификации общего и частного.

Наследование

- Иерархия классов представляется в виде древовидной структуры, в которой более общие классы располагаются ближе к корню, а более специализированные – на ветвях и листьях.
- Иногда предки называются *надклассами* или *суперклассами*, а потомки – *подклассами* или *субклассами*.

Полиморфизм

- *Полиморфизм* – возможность использовать в различных классах иерархии одно имя для обозначения сходных по смыслу действий и гибко выбирать требуемое действие во время выполнения программы.
- «один интерфейс — множество методов»
- Полиморфизм помогает уменьшить сложность программы, позволяя использовать один и тот же интерфейс для задания целого класса действий.

Классы

Разница подходов к составлению программ

- Процедурно-ориентированный язык:
  - последовательность операторов, выполняющих обработку данных определенного типа;
  - тип данных задается в операторе описания данных;
  - тип данных определяет:
    - формат представления данных в памяти компьютера;
    - типы операций и набор функций, которые могут использоваться с этими данными

Разница подходов к составлению программ

- Объектно-ориентированного подхода при составлении программ:
  - Средства языка позволяют определять не только данные, но и те операции, которые могут быть использованы с этими данными, т.е. есть эти средства определяют *тип данных, определяемый пользователем – абстрактные типы данных.*
  - Абстрактный тип данных используется для определения данных, которые называются **объектами.**
  - **Объект –** это совокупность данных, для которых определен набор функций. *Обработка этой совокупности данных может быть выполнена только с использованием этих функций.*

Класс

- В языке C++ для определения абстрактного типа используется понятие **класс.**
- Класс определяет структуру памяти будущего объекта по данным и те функции, которые будут обрабатывать эти данные.
- **Программист имеет возможность вводить собственные типы данных и определять операции над ними с помощью классов.**

Класс

- Определение класса включает в себя описание, из каких составных частей или **атрибутов** он состоит и какие операции (функции) опрелены для класса.
- Данные класса называются **полями** (по аналогии с полями структуры), а функции класса – **методами (**синонимы: *данные-члены* и *функции-члены***).**
- **Поля** и **методы** называются **элементами класса.**

Формат описания класса

```
class <имя>
        {
         [private: ]
              <описание скрытых элементов>
          public:
               <описание доступных элементов>
        };     // Описание заканчивается точкой с запятой
```

где **private** и **public –** спецификаторы доступа, управляющие видимостью элементов класса.

Формат описания класса

- Элементы, описанные после служебного слова **private,** видимы только внутри класса, т.е. они являются закрытыми и к ним может быть выполнено обращение только из членов-функций объекта. Этот вид доступа принят в классе *по умолчанию*.
- Если задан режим **public**, то это означает, что элементы объекта являются открытыми, и к ним может быть выполнено обращение как из членов-функций (полей) объекта, так и из внешней функции, в частности, из главной функции.

Формат описания класса

- Интерфейс класса описывается после спецификатора **public.**
- Действие любого спецификатора распространяется до следующего спецификатора или до конца класса.
- Можно задавать несколько секций **private** и **public,** порядок их следования значения не имеет.

Поля класса

- Могут иметь любой тип, кроме типа этого же класса (но могут быть указателями или ссылками на этот класс);
- Могут быть описаны с модификатором **const,** при этом они инициализируются только один раз (с помощью конструктора) и не могут изменяться;
- Могут быть описаны с модификатором **static,** но не **auto, extern, register.**
- Инициализация полей при описании не допускается.

Классы

- Классы могут быть:
  - **глобальными** (объявленными вне любого блока);
  - **локальными (**объявленными внутри блока, например, функции или другого класса**).**

Особенности локального класса

- внутри локального класса можно использовать типы, статические (**static**) и внешние (**extern**) переменные, внешние функции и элементы перечислений из области, в которой он описан;
- запрещается использовать автоматические переменные из этой области;
- локальный класс не может иметь статических элементов;
- методы этого класса могут быть описаны только внутри класса;
- если один класс вложен в другой класс, они не имеют каких-либо особых прав доступа к элементам друг друга и могут обращаться к ним только по общим правилам

Пример

- В программе необходимо оперировать комплексными числами. Комплексные числа состоят из вещественной и мнимой частей, и с ними можно выполнять арифметические операции.

```
class Complex {
    public:
            int real;                   // вещественная часть
            int imaginary;              //  мнимая часть
            void Add(Complex x);  // прибавить комплексное число
    };
```

Пример

- В этом примере определен класс **Complex,** представляющее комплексное число.
- Оно состоит из вещественной части – целого числа **real** и мнимой части, которая представлена целым числом **imaginary.** **real** и **imaginary –** это атрибуты класса.
- Для класса **Complex** определена одна операция или метод – **Add.**

Создание переменной класса

- Переменная типа **Complex:**

Complex number;

- Переменная с именем **number** содержит значение типа **Complex,** то есть содержит объект класса **Complex.**

Установка значений атрибутов объекта

Для существующего объекта возможна установка значений атрибутов объекта:

```
number.real=1;
number. imaginary=2;
```

Операция “.” обозначает обращение к атрибуту объекта.

Использование методов с объектами

```
Complex num2;
number.Add(num2);
```

- Метод **Add** выполняется с объектом.
- Методы часто называются сообщениями.
- Объекту **number** посылается сообщение **Add** с аргументом **num2.** Объект **number** принимает это сообщение и складывает свое значение со значением аргумента сообщения.

Примечания

- **Complex x1, x2, d;** // три объекта класса Complex
- **Complex dim [10];** // массив объектов класса Complex

Описание метода класса

```
Void Complex::Add(Complex x)
{
    this->real=this->real + x.real;
    this->imaginary=this-> imaginary + x. imaginary;
}
```

- Запись **this** говорит о том, что атрибут принадлежит к тому объекту, который выполняет метод **Add** (объекту, получившему сообщение **Add**).
- В большинстве случаев **this** можно опустить.
- В записи определения метода какого-либо класса упоминание атрибута класса без всякой дополнительной информации означает, что речь идет об атрибуте текущего объекта.

```
#include <iostream>
#include <conio.h>
using namespace std;
class Complex {
        public:
                int real;                  // вещественная часть
                int imaginary;          //  мнимая часть
                void Add(Complex x);  // прибавить комплексное число
        };

void Complex::Add(Complex x)
{
        real=real+x.real;
        imaginary=imaginary+x.imaginary;
}

int main()
{
        setlocale(LC_ALL, "rus");
        Complex number;
```

```
C:\Users\tsyganova\Documents\ex1.exe
Объект number, вещественная часть: 1, мнимая часть: 2
Объект num2, вещественная часть: 2, мнимая часть: 3
Объект number после выполения метода Add, вещественная часть: 3, мнимая часть: 5
```

Пример 2

Задана структура класса Q

- члены данные: массив целых чисел; n - переменная, определяющая текущий размер массива;
- члены-функции (методы):
  - функция ввода данных в объект;
  - функция вывода;
  - функция вычисления максимального элемента массива объекта;

Программа реализует следующий алгоритм:

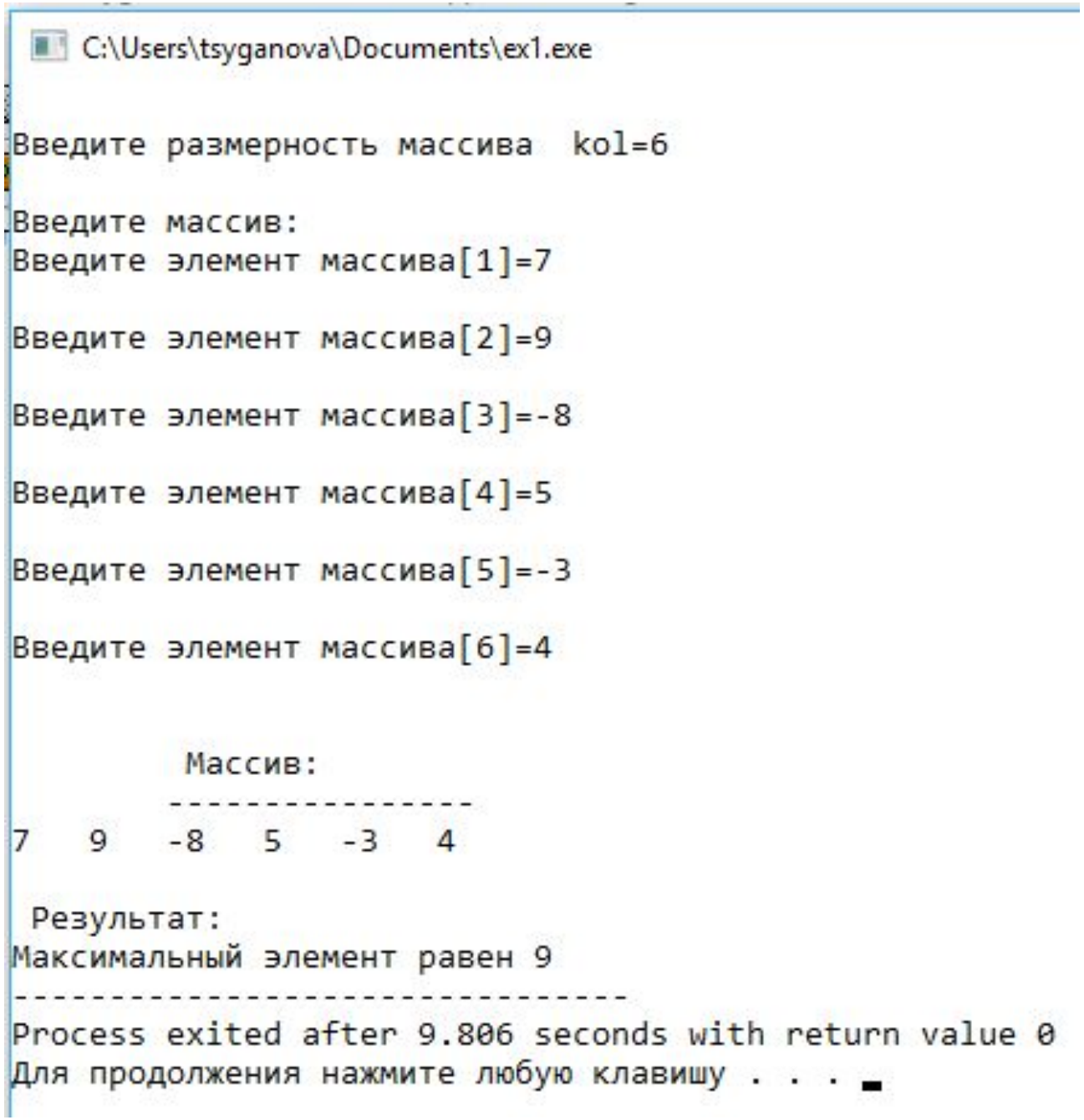
- создание объекта заданного класса;
- ввод данных в объект;
- вывод данных объекта;
- вычисление максимальной величины массива объекта
- печать результата.

```
#include <iostream.h>
#include <conio.h>
class Q        //Объявление класса  Q

//Объявление член-данных
      private:
            int mas[100];     //массив целых чисел;
            int kol;              //текущий размер массива

//Объявление член-функций (методов)
      public:
            void Enter();      /*Объявление член-функции ввода данных  */
                  void output();  /*Объявление член-функции вывода данных */
            int funk();          /*объявление член-функции нахождения
                                       максимального элемента массива */
};                  //конец объявления класс Q
```

```
int main()
{
     setlocale(LC_ALL, "rus");
    int m;
    Q obj;                    //Создание объекта
    obj.Enter();              //Ввод данных
    obj.output();             //Вывод данных
    m=obj.funk();             //Нахождение max
    cout <<"\n\n Результат:"
        <<"\Максимальный элемент равен: " <<m;
    return 0;
}
```

```
void Q::Enter()
//отложенное определение функции ввода данных
{
	cout <<"\nВведите размер массива  kol=";
	cin  >>kol;
	cout <<"\nВведите массив чисел:\n";
	for(int i=0;i<kol;i++)
	{
		cout <<«Введите mas["<<(i+1)<<"]=";
		cin  >>mas[i];
		cout <<endl;
	}
}
```

```
void Q::output()  /* отложенное определение функции вывода массива объекта   */
```


C:\Users\tsyganova\Documents\ex1.exe
Введите размерность массива kol=6
Введите массив:
Введите элемент массива[1]=7
Введите элемент массива[2]=9
Введите элемент массива[3]=-8
Введите элемент массива[4]=5
Введите элемент массива[5]=-3
Введите элемент массива[6]=4
Массив:
7 9 -8 5 -3 4
Результат:
Максимальный элемент равен 9
Process exited after 9.806 seconds with return value 0
Для продолжения нажмите любую клавишу . . .

Пример 3

Программа реализует следующий алгоритм:

- создание объекта с выделенной памятью для массива слов ;
- ввод массива слов в объект;
- вывод данных объекта;
- выполнение функции вычисления слова с max длиной;
- распечатка в основной программе найденного слова и его длины с использованием возвращенных параметров - слова и его длины.

Задана структура класса Q:

- члены данные:
- массив слов (максимальный размер массива 100);
- максимальная длина слова len=11;
- n - количество элементов массива (текущий размер);

Члены функции (методы):

- функция ввода данных объекта;
- функция вывода данных объекта;
- функция определения слова с максимальной длиной (передача этого слова в вызывающую функцию и его порядкового номера).

```
#include <iostream.h>
#include <string.h>
#include <conio.h>
class Q                    //Объявление класса Q
{
    int kol;               //текущий размер массива  слов
    char mas[100][11];  //массив слов
    public:          //Объявление член-функций (методов)
    void Enter();      //Объявление член-функции ввода данных
    void Output();   //Объявление член-функции вывода данных
    int Funk(char *);    /*объявление член-функции нахождения слова максимальной длины   и длины этого слова*/
};                 //конец объявления класса Q
```

```
int main()
{
    setlocale(LC_ALL, "rus");
    int max=NULL;       // результат- максимальная длина и слово максимальной длины
    char str[255]={NULL};
    Q obj;              //Объявление объекта
    obj.Enter();    //Ввод данных
    obj.Output();   //Вывод данных
    max=obj.Funk(str);    /*Вычисление  слова с максимальной длиной и запись слова в строку str и длины в переменную max  */
cout <<"\n  Результат:"\nСамое длинное слово:"<<str<<"\nего длина: "<<max;
        return 0;
}
```

```
void Q::Enter() /* отложенное определение член-функции ввода данных
*/
{
    cout <<"\nВведите размер массива kol=";
    cin >>kol;
    cout <<"\n Ввод массива слов:”;
    cin.get();
    for(int i=0; i<kol; i++)  //цикл ввода массива слов
    {
        cout <<"\nВведите mas["<<(i+1)<<"] :”;
        cin.getline(mas[i],11);
        if(!cin) cin.clear();
    }
}
```

```
void Q::Output()        /* отложенное определение  член-функция
 вывода данных     */
{
        cout <<"\n\n Вывод массива слов:"
                <<"\n----------------------";
        for(int i=0; i<kol; i++)
                cout <<"\n["<<(i+1)<<"]. ->"<<mas[i];
}
```

```
C:\Users\tsyganova\Documents\ex1.exe
======================

Введите размер массива kol=5

 Ввод массива слов:
--------------
Введите mas[1] :qqqqq

Введите mas[2] :sss

Введите mas[3] :aaaaaaa

Введите mas[4] :dd

Введите mas[5] :s


 Вывод массива слов:
----------------------
[1]. ->qqqqq
[2]. ->sss
[3]. ->aaaaaaa
[4]. ->dd
[5]. ->s
  Результат:
--------------
Самое длинное слово:aaaaaaa
его длина: 7
```

```
ределение член-функция определения слова с max длиной   */
ax_word) // возвращает слово и длину
```

Переопределение операций

Переопределение операций

- В языке C++ допустимо, что класс будет практически неотличим от предопределенных встроенных типов при использовании в выражениях.
- Для класса можно определить операции сложения, умножения и т.д. пользуясь стандартной записью таких операций, т.е. **x + y.**
- В языке C++ считается, что подобная запись – это также вызов метода с именем **operator+** того класса, к которому принадлежит переменная **x**.

Переопределение операций

```
// определение класса комплексных чисел
class Complex
{
    public:
            int real;                   // вещественная часть
            int imaginary;       //  мнимая часть
        // прибавить комплексное число
    Complex operator+ (const Complex x)   const;
    };
```

Complex operator+ (const Complex x) const;

- Вместо метода **Add** появился метод **operator+.**
- Этот метод возвращает значение типа **Complex** (операция сложения в результате дает новое значение того же типа, что и типы операндов).
- Перед аргументом метода появилось ключевое слово **const**, означающее, что при выполнении данного метода аргумент изменяться не будет.
- Второе ключевое слово **const** означает, что объект, выполняющий метод, не будет изменен.
- При выполнении операции сложения **x + y** над двумя величинами **x** и **y** сами эти величины не изменяются.

Определение операции сложения

```
Complex:: operator+ (const Complex x) const
{
    Complex result;
    result.real = real + x.real;
    result. imaginary = imaginary + x. imaginary;
    return result;
}
```

Переопределение операций

- В языке C++ допускается определение в одном классе нескольких методов с одним и тем же именем, но разными типами и количеством аргументов.
- Определение методов или атрибутов с одинаковыми именами в разных классах не вызывает проблем, поскольку пространства имен разных классов не пересекаются.

Переопределение операций

```
// определение класса комплексных чисел
class Complex
{
    public:
        int real;              // вещественная часть
        int imaginary;     //  мнимая часть
        // прибавить комплексное число
      Complex operator+ (const Complex x)   const;
        // прибавить целое число
        Complex operator+ (long x)   const;
};
```

Пример

```
Complex c1;
Complex c2;
long x;
c1 + c2;
c2 + x;
```

Полный пример

```
#include <iostream>
#include <conio.h>
using namespace std;
class Complex {
        public:
              int real;                  // вещественная часть
              int imaginary;          //  мнимая часть
              // прибавить комплексное число
        Complex operator+(const Complex x)const;
             // прибавить целое число
             Complex operator+(long x)const;

        };



Complex Complex::operator+(const Complex x) const
{
```

```
    setlocale(LC_ALL, "rus");
    Complex c1;
    c1.real=1;
    c1.imaginary=2;
    cout<<"Объект c1, вещественная часть: "<<c1.real<<", мнимая часть: "<<c1.imaginary;
    Complex c2;
    c2.real=2;
    c2.imaginary=3;
    cout<<"\nОбъект num2, вещественная часть: "<<c2.real<<", мнимая часть: "<<c2.imaginary;
    long x=10;
    Complex c3;
    c3=c1+c2;
    cout<<"\nОбъект c3 после выполнения метода c1+c2, вещественная часть: "<<c3.real<<", мнимая часть: "<<c3.imaginary;
    c2=c2+x;
    cout<<"\nОбъект c2 после выполнения метода c2+x, вещественная часть: "<<c2.real<<", мнимая часть: "<<c2.imaginary;
    return 0;
}
```