**ЛАБОРАТОРНАЯ РАБОТА 1**

**ИССЛЕДОВАНИЕ ОПИСАНИЯ И ИСПОЛЬЗОВАНИЯ КЛАССОВ**

**Цель лабораторной работы -** исследовать описание и использование класса.

**Задание.**

1. Изучить способы описания классов в С++.
2. Написать программу на языке С++ для исследования описания и использования класса (табл. 7).
3. Написать и отладить программу на языке С++, которая демонстрирует случаи применения классов, приведенные в табл.6.

Перед выполнением заданий необходимо ознакомиться с методическими указаниями и примечаниями представленными в конце документа.

Таблица 6

**Варианты заданий**

| *№* | *Описание класса* | | |
| --- | --- | --- | --- |
| *Класс* | *Элементы* | *Задание* |
|  | Треугольник | Координаты вершин, конструктор с параметрами, деструктор, методы вычисления площади и периметра, метод вывода треугольника | Описание объекта, вычисление и вывод его площади и периметра, вывод объекта |
|  | Вектор | Координата конца вектора, конструктор с параметрами, деструктор, методы вычисления полярных координат, метод вывода вектора | Описание объекта, вычисление и вывод его полярных координат, вывод объекта |
|  | Строка | Цифровые символы, конструктор с параметрами, деструктор, метод вычисления длины, метод обращения строки, метод вывода строки | Описание объекта, вывод его длины, вывод объекта |
|  | Прямоугольник | Координаты левой верхней и правой нижней вершин, конструктор с параметрами, деструктор, методы вычисления площади и периметра, метод вывода прямоугольника | Описание объекта, вычисление и вывод его площади и периметра, вывод объекта |
|  | Отрезок | Координаты начала и конца, конструктор с параметрами, деструктор, методы вычисления длины и угла между осью ОХ, метод вывода отрезка | Описание объекта, вычисление и вывод его длины и угла с осью ОХ, вывод объекта |
|  | Строка | Буквенные символы, конструктор с параметрами, деструктор, метод вычисления длины, метод сортировки строки по возрастанию, метод вывода строки | Описание объекта, вывод его длины, вывод объекта |
|  | Квадрат | Координаты левой верхней и правой нижней вершин, конструктор с параметрами, деструктор, методы вычисления площади и периметра, метод вывода квадрата | Описание объекта, вычисление и вывод его площади и периметра, вывод объекта |

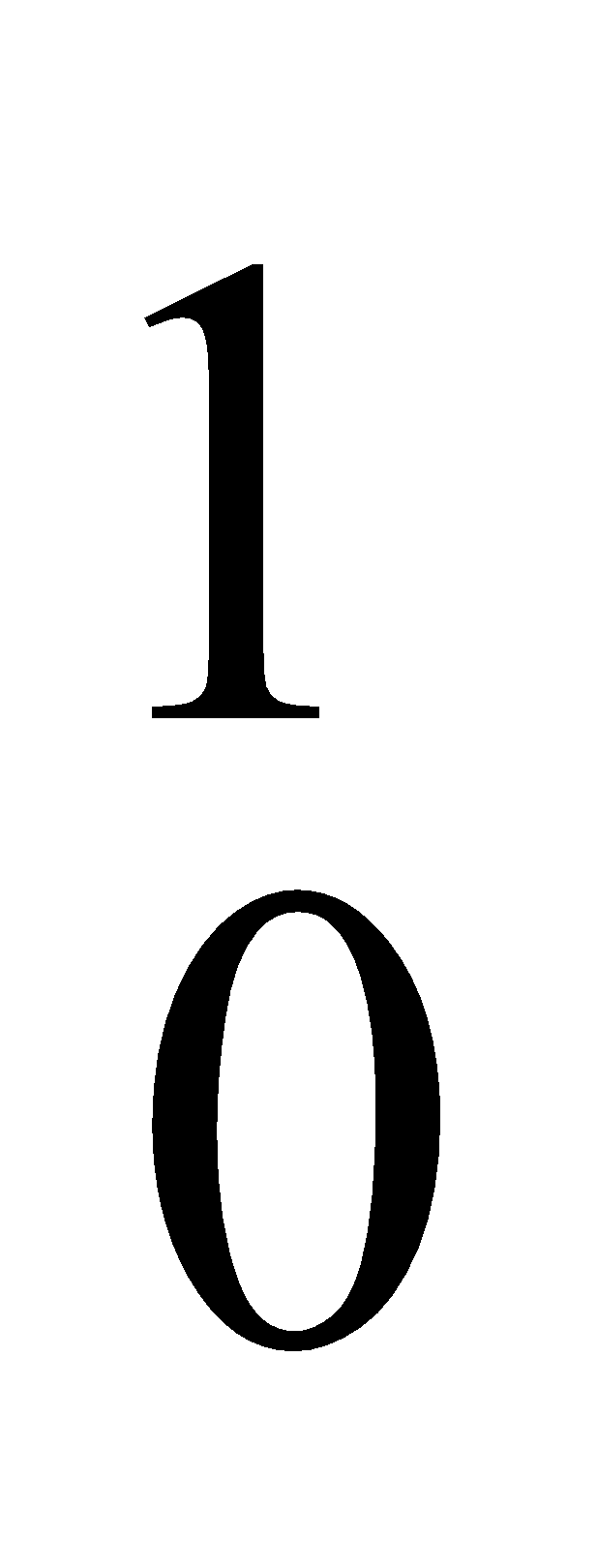
Окончание табл. 6

|  | Вектор | Полярные координаты (длина и угол), конструктор с параметрами, деструктор, методы вычисления координат конца вектора, метод вывода вектора | Описание объекта, вычисление и вывод его координат конца вектора, вывод объекта |
| --- | --- | --- | --- |
|  | Строка | Специальные символы, конструктор с параметрами, деструктор, метод вычисления длины, метод сортирования строки по убыванию, метод вывода строки | Описание объекта, вывод его длины, вывод объекта |
|  | Трапеция | Координаты вершин, конструктор с параметрами, деструктор, методы вычисления площади и периметра, метод вывода трапеции | Описание объекта, вычисление и вывод его площади и периметра, вывод объекта |
|  | Отрезок | Координаты начала и конца, конструктор с параметрами, деструктор, методы вычисления длины и угла между осью ОY, метод вывода отрезка | Описание объекта, вычисление и вывод его длины и угла с осью ОХ, вывод объекта |
|  | Строка | Цифровые символы, конструктор с параметрами, деструктор, метод вычисления длины, метод удаления с строки символа ’5', метод вывода строки | Описание объекта, вывод его длины, вывод объекта |
|  | Ромб | Координаты вершин, конструктор с параметрами, деструктор, методы вычисления площади и периметра, метод вывода ромба | Описание объекта, вычисление и вывод его площади и периметра, вывод объекта |
|  | Строка | Буквенные символы, конструктор с параметрами, деструктор, метод вычисления длины, метод обмена символа ’a' на символы '++’, метод вывода строки | Описание объекта, вывод его длины, вывод объекта |
|  | Строка | Буквенные символы, конструктор с параметрами, деструктор, метод вычисления длины, метод циклического сдвига строки в правую сторону на один символ, то есть последний символ на первое место, метод вывода строки | Описание объекта, вывод его длины, вывод объекта |

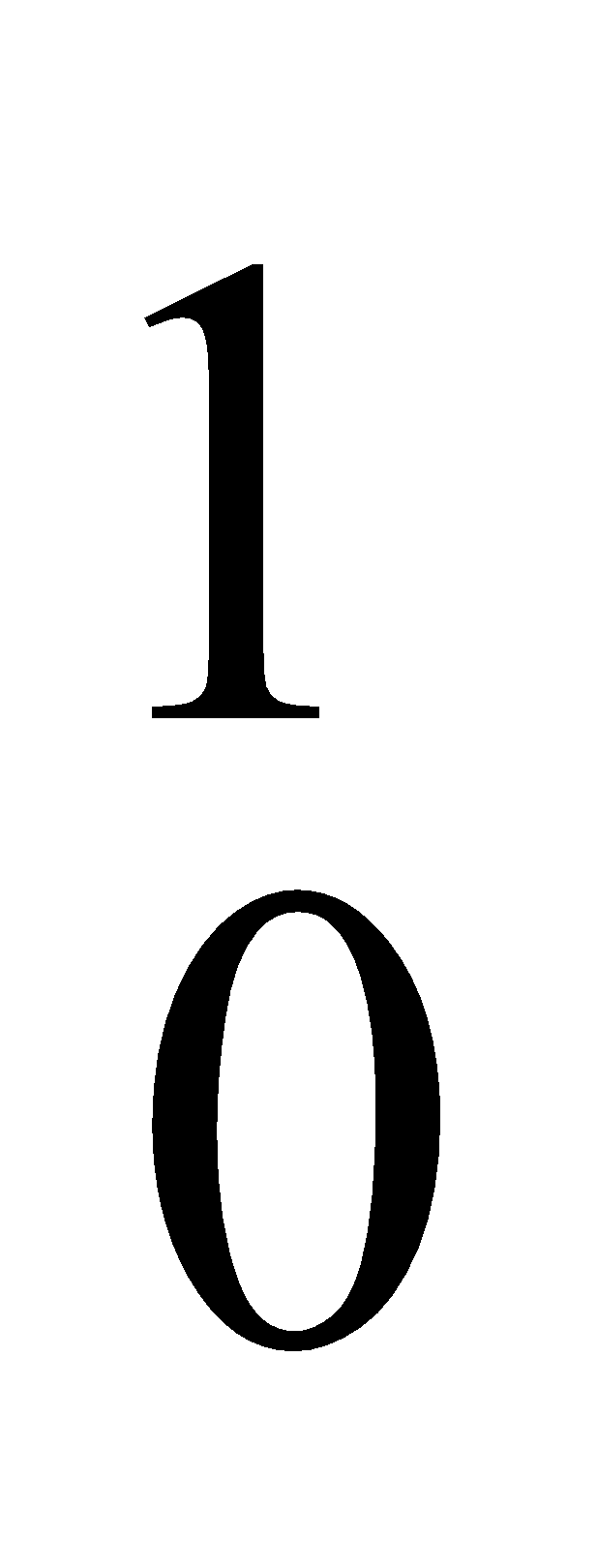
**Теоретические сведения**

**Описание класса:**

<описание класса> ::= class <пометка класса> {<тело класса>};

<тело класса> ::= {private:} <закрытый элемент>

public: <открытый элемент>

{protected: <защищенный элемент> }

<пометка класса> ::= <идентификатор>

<закрытый элемент> ::= <описание переменной> | <описание функции> | <описание прототипа функции> |

<описание статической переменной>

<описание статической переменной> ::= static <описание переменной>

<открытый элемент> ::= <конструктор> | <деструктор> | <описание переменной> | <метод> |

<описание прототипа метода> | <метод типа static> | <метод типа const>

<защищенный элемент> ::= <описание переменной> | <описание функции> | <описание прототипа функции>

<конструктор> ::= <пометка класса> ({<список формальных параметров>}0)

{<тело конструктору>}

<тело конструктору> ::= <тело закрытой подпрограммы>

<деструктор> ::= ~<пометка класса> ( ) {<тело деструктору>}

<тело деструктору> :: = <тело закрытой подпрограммы>

<метод> ::= <описание функции>

<метод типа static> ::= static <метод>

<метод типа const> ::= <описание прототипа метода> const

**Описание метода вне класса:**

<Описание метода вне класса> ::=

<описание метода> | <описание метода типа static> | <описание метода типа const>

<описание метода> ::= <пометка типа возвращаемого значения>

<пометка класса>::<пометка метода>({<список формальных параметров>}0)

<тело метода>

<пометка метода> ::= <идентификатор>

<тело метода> ::= {{<операторы>}0}

<описание метода типа static> ::= <описание метода>

<описание метода типа const> ::= <пометка типа возвращаемого значения>

<пометка класса>::<пометка метода>({<список формальных параметров>}0) const

<тело метода>

**Описание конструктора и деструктора вне класса:**

<Описание конструктора вне класса> ::= <пометка класса>::<конструктор>

<Описание деструктора вне класса> ::= <пометка класса>::~<деструктор>

**Описание статических элементов вне класса:**

<пометка типа> <пометка класса> :: <идентификатор> = <статическое выражение>

**Использование конструкторов, методов и статических элементов:**

<Использование конструкторов> ::= <пометка класса> <пометка объекта>;

<пометка объекта> ::= <идентификатор> | <идентификатор>({<список фактических параметров>}) |

&<идентификатор> | \*<идентификатор> | <идентификатор>[<R-выражение>]

<Использование методов> ::= <идентификатор объекта>.<вызов метода> |

<идентификатор объекта>-><вызов метода>

<вызов метода> ::= <вызов функции>

<Использование метода типа const> ::= <Использование методов>

<Использование метода типа static> ::= <пометка класса> :: <вызов метода>

**Методические указания**

Описание класса может составляться с трех частей: закрытой, открытой и защищенной. По умолчанию первой частью есть закрытая. Поэтому рекомендуется начинать описание класса с закрытых элементов.

Для наглядности описания класса пользователю рекомендуется описание таких открытых элементов, как <конструктор>, <деструктор> и <метод>, приводить вне класса. Помните, если в классе описан статический элемент, то его необходимо обязательно определить и инициализировать вне класса.

Все методы класса, кроме <методов типа static>, при вызове получают неявный аргумент (указатель this – указатель на сам класс).

Пользователю в роботе необходимо самостоятельно привести описание всех типов конструкторов и деструкторов, несмотря на то, что в языке Си++ конструктор по умолчанию и деструктор создаются автоматически.

Таблица 7

| Описание класса | | | | | | Использование класса | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| описание частей | | | | описание методов в классе | описание методов вне класса | Использование конструкторов | использование методов класса |
| закрытые элементы | защищенные  элементы | Открытые  элементы | |
| конст-руктор | дестру-ктор |

**Примечания**

Для решения задач требующих вывода графических объектов рекомендуется определить в вашем классе методы инициализации и закрытия графического режима (возможно с модификатором static например:

#include <stdio.h>

#include <graphics.h>

#include <stdlib.h>

class CMyClass

{

public:

…………………….

private:

//методы инициализации и закрытия графического режима

static void InitGraphMode();

static void CloseGraphMode();

//ваши данные

…………………….

};

//Метод инициализации графического режима

void CMyClass::InitGraphMode()

{

/\* request auto detection \*/

int gdriver = DETECT, gmode, errorcode;

/\* initialize graphics mode \*/

initgraph(&gdriver, &gmode, "");

/\* read result of initialization \*/

errorcode = graphresult();

if (errorcode != grOk) /\* an error occurred \*/

{

printf("Graphics error: %s\n", grapherrormsg(errorcode));

printf("Press any key to halt:");

getch();

exit(1); /\* return with error code \*/

}

}

//метод закрытия графического режима

void CMyClass::CloseGraphMode()

{

closegraph();

}

В методах вывода на экран графических объектов необходимо использовать функции аналогичные функциям определенным в модуле Graph языка Pascal, такие как line, circle, putpixel, move, moveto и т.д. Для выполнения задач с графикой рекомендуется текстовые данные отображать в текстовом режиме, а затем после нажатия пользователем какой-либо клавиши выводить на экран графическое представление объекта.

В текстовом режиме для вывода данных на экран разрешается использовать любые функции ввода/вывода.

В классах работающих со строками необходимо в конструкторе создавать копию задаваемой в качестве параметра конструктора строки например

//секция описания

class MyString

{

public:

MyString(char\* new\_str);

virtual ~MyString();

………………………

protected:

char\* m\_data;

};

//секция реализации

MyString::MyString(char\* new\_str):m\_data(NULL)

{

//если в качестве параметра задана не пустая строка

//делаем ее копию и сохраняем указатель на новую строку в m\_data

if (new\_str)

m\_data=strdup(new\_str);

}

//деструктор освобождает память занимаемую строкой

MyString::~MyString()

{

free(m\_data);

}

В приведенном выше примере использована стандартная функция strdup которая вызывает функцию malloc для выделения необходимого объема памяти а затем копирует в нее строку указанную в качестве параметра. В деструкторе необходимо не забыть удалить память выделенную в strdup, для этого необходимо вызвать стандартную функцию free. Также для решения этой же задачи возможен другой подход:

//секция реализации

MyString::MyString(char\* new\_str):m\_data(NULL)

{

//если в качестве параметра задана не пустая строка

//делаем ее копию и сохраняем указатель на новую строку в m\_data

if (new\_str) {

m\_data=new char[strlen(new\_str)+1];

strcpy( m\_data, new\_str);

}

}

//деструктор освобождает память занимаемую строкой

MyString::~MyString()

{

delete[] m\_data;

}

В указанном фрагменте применен подход в стиле С++ когда для выделения памяти используется оператор new при этом используется форма оператора new для инициализации массива при этом в качестве параметра необходимо задать объем требуемой памяти который равен размеру исходной строки + 1 символ для размещения символа \0 окончания строки. В деструкторе при этом используется форма оператора delete[] используемая для удаления массива.

Необходимо помнить также что строки в языках С/С++ представляют собой массив символов типа char. Поэтому для работы со строками возможна использовать операции для работы с массивами. Приведем пример:

char\* new\_str;

Запись означает указатель на строку т.е. данная переменная может содержать начальный адрес строки символов или указывать на первый элемент строки символов. Присваивание строки можно выполнить следующим образом:

new\_str=”Ваша строка”;

При этом переменная указатель будет содержать адрес по которому в оперативной памяти содержится буква ‘В’. Для доступа к ячейке памяти используется операция разыменования указателя \*. Например:

\*new\_str=’R’;

Данный оператор произведет замену буквы В на букву R обратите внимание что при этом буква R указывается в одинарных кавычках. Аналогично можно изменить любую букву в строке при этом надо помнить что все последующие буквы имеют адрес на единицу больше чем предыдущая буква например для доступа к букве ‘а’ можно использовать запись вида

\*(new\_str+1)=’п’;

Также необходимо помнить, что нельзя обращаться к памяти за пределами строки символов например если указать **\*(new\_str+25)=’п’;** произойдет ошибка в программе. Для прохода по строке часто используется операция инкремента указателя new\_str++ теперь указатель уже указывает не на первую а на вторую букву т.е. если записать

\*new\_str=’R’;

то мы заменим уже не первую а вторую букву аналогично для

\*(new\_str+1)=’п’;

мы заменим уже не втрорую а третью букву.

Основным правилом при работе со строками а также и с массивами других типов является то, что для доступа к строке(массиву) можно использовать как операцию разыменования указателя так и операцию доступа к элементам массива т.е. если принять что в выше указанном примере new\_str указывает на начало строки то записи:

\*new\_str=’R’;

и

new\_str[0]=’R’;

эквивалентны!!! Аналогично

\*(new\_str+1)=’п’;

и

new\_str[1]=’п’;

тоже эквивалентны и при этом большинство современных компиляторов генерируют одинаковый код на языке ассемблера так, что выбор используемой операции с программной точки зрения не имеет значения. Все выше приведенное также справедливо и для указателей-массивов других стандартных типов данных например int. Т.е. имея запись

int a[10];

Доступ ко втрому символу массива осуществляется как

\*(a+1)=5;

или

a[1]=5;

Необходимо помнить, что номера элементов массива в Си начинаются с 0 а не с 1.

Все выше указанные особенности необходимо применять в задачах для замены символов в строках а также анализа содержимого строк.

**Для реализации метода отыскания длины строки вы не должны использовать стандартную функцию strlen. Вам необходимо осуществить проход по символам строки использовав указанный выше подход и сосчитать все символы до тех пор пока не будет достигнут символ окончания строки \0 т.е.**

**if new\_str[i]==’\0’ означает конец строки выход из цикла.**

В задачах где необходимо удалить символ со строки необходимо найти позицию указанного символа и осуществить сдвиг оставшейся части строки (стоящей справа) на один символ влево включая конец строки.

В задачах где требуется заменить один символ на два символа необходимо увеличить размер памяти занимаемой строкой на один символ это можно сделать использовав стандартную функцию realloc которая имеет следующий синтаксис

**void** **\*realloc(void** **\****memblock***,** **size\_t** *size* **);**

где *memblock* указатель на блок памяти размер которого необходимо изменить *size* новый размер блока, тип **size\_t** эквивалент типа int. Функция возвращает указатель на новый (увеличенный или уменьшенный) блок памяти т.е. если new\_str есть указатель на строку для увеличения строки на 1 байт можно записать

new\_str=**realloc(** new\_str**,** strlen(new\_str)+1+1**);**

где strlen(new\_str)+1 исходный размер строки

strlen(new\_str)+1+1 новый размер строки

Далее необходимо сдвинуть всю часть строки стоящую справа от искомого заменяемого символа на один символ вправо а затем заменить сам символ и символ стоящий справа на новый символ.

**Контрольные вопросы**

1. Что такое класс?
2. Рассказать с каких частей составляется описание класса.
3. Что такой конструктор и деструктор?
4. Приведите примеры типов конструкторов.
5. Для чего используется конструктор?
6. Для чего используется деструктор?
7. Приведите примеры использования методов.