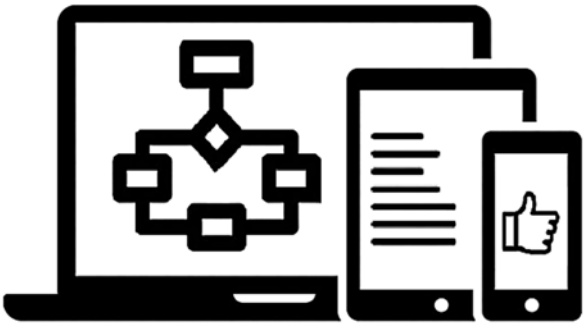
МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение информационных технологий»

# СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

***Методические рекомендации к лабораторным работам для студентов направления подготовки 09.03.01 «Информатика и вычислительная техника» очной формы обучения***



Могилев 2023

УДК 004.42
ББК 32.973-018
С40

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Программное обеспечение информационных технологий» «8» августа 2023 г., протокол № 1

Составитель канд. техн. наук, доц. Н. Н. Горбатенко

Рецензент канд. техн. наук, доц. И. В. Лесковец

Методические рекомендации разработаны на основе рабочей программы по дисциплине «Системное программирование» для студентов направления подготовки 09.03.01 «Информатика и программирование» очной формы обучения и предназначены для использования при выполнении лабораторных работ.

Учебное издание

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

| | |
|---|---|
| Ответственный за выпуск | В. В. Кутузов |
| Корректор | И. В. Голубцова |
| Компьютерная верстка | Н. П. Полевничая |

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 21 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

# Содержание

# Введение

Методические рекомендации разработаны на основе рабочей программы по дисциплине «Системное программирование» для студентов направления подготовки 09.03.01 «Информатика и программирование» очной формы обучения и предназначены для использования при выполнении лабораторных работ.

Цель преподавания дисциплины «Системное программирование» – получение навыков разработки программ на языке C/C++ с использованием процедурной и объектно-ориентированной парадигмы программирования.

Основные правила выполнения лабораторных работ:

– ознакомиться с целью предстоящей работы и условием индивидуального задания;

– изучить вопросы раздела «Теоретические сведения», используя указанные литературные источники и конспект лекций;

– продумать алгоритм решения задачи индивидуального задания;

– набрать и отладить текст программы в среде Visual Studio.NET;

– подготовить отчёт по лабораторной работе (рисунок 1.1);

– защитить лабораторную работу. Защита включает в себя демонстрацию работы программы преподавателю и ответы на его вопросы по теме лабораторной работы в объёме методических рекомендаций. После защиты лабораторной работы преподаватель ставит на титульном листе свою подпись и дату. Только после этого лабораторная работа считается полностью выполненной и студент может приступить к выполнению следующей работы.

Белорусско-Российский университет
Кафедра «Программное обеспечение информационных технологий»

Отчёт по лабораторной работе №__
по дисциплине «Программирование»

Название темы лабораторной работы
Вариант задания –

| | |
|---|---|
| Выполнил | Проверил |
| Ст. гр. АСОИР-221 | Преподаватель |
| ФИО | ФИО |

Могилев 2023

Рисунок 1.1 – Структура титульного листа

Отчёт по лабораторной работе оформляется на листах формата А4 и должен содержать:

– титульный лист;
– цель работы;
– краткие теоретические сведения;
– текст индивидуального задания;
– листинг разработанной программы;
– скриншоты с результатами выполнения программы;
– выводы о проделанной работе.

# 1 Лабораторная работа № 1. Основные элементы языка C++. MS Visual Studio 2019

**Цель работы**: изучение среды программирования Microsoft Visual Studio 2019; получение навыков создания консольных приложений и программирования линейных алгоритмов.

## *1.1 Теоретические сведения*

Программа, создаваемая в среде Visual Studio , всегда оформляется в виде отдельного проекта. Проект (project) – это набор взаимосвязанных исходных файлов, предназначенных для решения определенной задачи, компиляция и компоновка которых позволяет получить выполняемую программу. В проект входят как файлы, непосредственно создаваемые программистом, так и файлы, которые автоматически создает и редактирует среда программирования.

Для создания нового проекта необходимо:

– в стартовом окне Visual Studio выбрать Create a new project;

– в появившемся окне выбрать язык программирования C++ и среди шаблонов проектов выбрать тип проекта Console App;

– в следующем окне в поле Project name ввести имя проекта, а в поле Location указать расположение проекта и нажать на Create для создания проекта.

После этого Visual Studio создаст типовой проект консольного приложения на C++. Справа в окне Solution Explorer отображается структура проекта. В реальности окно Solution Explorer содержит решение. В данном случае оно называется так же, как имя проекта. Решение может содержать несколько проектов. В проекте есть ряд каталогов: External Dependencies – содержит файлы, которые используются в файлах исходного кода, но не являются частью проекта; Header Files – предназначен для хранения заголовочных файлов с расширением .h; Resource Files – предназначен для хранения файлов ресурсов; Source Files – хранит файлы с исходным кодом проекта.

Каталог Source Files содержит файл (название проекта + расширение фай0 ла .cpp). Этот файл видим в текстовом редакторе Visual Studio. По умолчанию он содержит следующий код:

```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "Hello World!\n";
6  }
```

Здесь iostream – стандартный заголовочный файл библиотеки ввода-вывода. Эта библиотека содержит информацию о потоке cout, используемом в данной программе. #include является директивой препроцессора, заставляющей включить в программу текст из заголовочного файла iostream.

Функция main() – эта главная функция программы. Она вызывается операционной системой при запуске программы. Фигурные скобки { } отмечают начало и конец тела функции main().

В теле функции main() определена инструкция std::cout << "Hello, world!\n", которая выводит в окно консоли сообщение Hello, world!.

cout – это выходной поток, направленный на терминал, << – оператор вставки данных (строки ″Hello, world!″) в выходной поток.

Префикс std:: указывает, что объект cout определен в стандартном пространстве имен std.

Для компиляции и запуска программы нужно нажать сочетание клавиш Ctrl + F5 или выбрать пункт меню Debug –> Start Without Debugging. В итоге Visual Studio передаст исходный код программы компилятору, который создаст из кода исполняемый файл .exe, который будет запущен на выполнение.

Более подробные теоретические сведения приведены в [1, с. 5–13].

### *1.2 Индивидуальное задание*

Вычислить значение выражения при заданных исходных данных. Сравнить полученное значение с указанным правильным результатом.

1 $s = \dfrac{2\cos\left(x - \dfrac{2}{3}\right)}{\dfrac{1}{2} + \sin^2 y}\left(1 + \dfrac{z^2}{3 - z^2/5}\right).$

При $x = 14{,}26$, $y = -1{,}22$, $z = 3{,}5 \cdot 10^{-2}$ ответ $s = 0{,}749155$.

2 $s = \dfrac{\sqrt[3]{9 + (x-y)^2}}{x^2 + y^2 + 2} - e^{|x-y|}\mathrm{tg}^3 z.$

При $x = -4{,}5$, $y = 0{,}75 \cdot 10^{-4}$, $z = -0{,}845 \cdot 10^{-2}$ ответ $s = -3{,}23765$.

3 $s = \dfrac{1 + \sin^2(x+y)}{\left|x - \dfrac{2y}{1 + x^2y^2}\right|} x^{|y|} + \cos^2\left(\mathrm{arctg}\dfrac{1}{z}\right).$

### *Контрольные вопросы*

1 Назовите основные этапы создания консольного приложения в среде программирования Visual Studio.

2 Какова структура программы на языке C++?

3 Для чего необходима директива препроцессора #include?

# 2 Лабораторная работа № 2. Ввод и вывод информации

**Цель работы**: получение навыков ввода-вывода информации при программировании на языке C++.

## *2.1 Теоретические сведения*

Описание объектов для управления вводом-выводом содержится в заголовочном файле iostream. При подключении этого файла с помощью директивы препроцессора #include <iostream> в программе автоматически создаются объекты-потоки – cin для ввода с клавиатуры и cout для вывода на экран, а также операции помещения в поток << и чтения из потока >>.

С помощью объекта cin>> можно присвоить значение любой переменной. Например, если переменная i объявлена как целочисленная, то инструкция cin>> i; означает, что в переменную i будет записано некое целое число, введенное с клавиатуры Если нужно ввести несколько переменных, следует написать cin>>x>>y>>z;.

Объект cout и операция << позволяют вывести на экран значение любой переменной или текст. Текст необходимо заключать в двойные кавычки. Кроме того, допустимо применение символов управляющих последовательностей \t и \n. Инструкция cout<<i; означает вывод на экран значения переменной i, а инструкция cout<<x<<"\t"<<y; выведет на экран значения переменных x и y, разделенных символом табуляции.

## *2.2 Индивидуальное задание*

Создать проект в Visual Studio и написать код, в котором:
– вводятся значения в переменные целого типа x и y;
– вычисляется сумма, разность, произведение и отношение этих переменных;
– выводится результаты вычислений в виде

Сумма x+y =
Разность x – y =
Произведение x*y=
Отношение x/y=

Отладить и запустить проект.

### *Контрольные вопросы*

1 Каким образом можно осуществить вывод информации на экран?
2 Как можно организовать ввод данных с клавиатуры?

# 3 Лабораторная работа № 3. Разветвляющие программы

**Цель работы**: получение навыков написания разветвляющихся программ.

## *3.1 Теоретические сведения*

Оператор условной передачи управления if. Формат оператора выбора

```
if (логическое выражение) оператор 1;
  else оператор 2;
```

Если логическое выражение истинно, то выполняется оператор 1, иначе – оператор 2.

Оператор множественного выбора switch. Общая форма оператора следующая:

```
switch(переменная выбора) {
  case const 1: операторы 1 ;   break;
                                    …
  case const N: операторы N;   break;
  default:          операторы N+1;
}
```

При использовании оператора switch сначала анализируется переменная выбора и проверяется, совпадает ли её значение со значением одной из констант. При совпадении выполняются операторы этого case. Конструкция default (может отсутствовать) выполняется, если результат выражения не совпал ни с одной из констант.

***Пример*** – Написать программу нахождения действительных корней квадратного уравнения общего вида ax 2 + bx + c = 0.

```
#include <iostream>
#include <math.h>
using namespace std;
int main ()
{
	int a, b, c; // Коэффициенты a, b, c
	setlocale (LC_ALL, "Russian"); // подключение русификатора
	cout << "Введите коэффициенты a b c > ";
	cin >> a >> b >> c; //Ввод данных
	double d=b*b-4*a*c; //Вычисление дискриминанта
	if ( d>=0) //Если дискриминант больше или равен 0,
	{
		//Вычислить корни x1, x2
		double x1= (-b+sqrt (d))/ (2.0*a);
		double x2= (-b-sqrt (d))/ (2.0*a);
```

```
            //Вывод на экран значений корней:
            cout << "Первый корень = " << x1 << endl;
            cout << "Второй корень = " << x2 << endl;
        }
        //Если корней нет, то вывод сообщения "Корней нет"
        else cout << "Корней нет" << endl;
}
```

### *3.2 Индивидуальные задания*

1 Определить, какое из выражений является меньше: a*ln(b) или $\sin(a)*b^2$.

2 Даны произвольные числа a, b, c. Если нельзя построить треугольник с такими длинами сторон, то выдать соответствующее сообщение; если можно, то вычислить его площадь по формуле Герона.

3 Пользуясь оператором switch, по введенному номеру месяца выдать на экран сообщение о времени года и названии введенного месяца. Например: 1 – январь, зима … .

### *Контрольные вопросы*

1 Какие операторы позволяют организовать выбор между несколькими вариантами?

2 Что понимают под логическим выражением?

3 Чем отличается логическое И от логического ИЛИ?

4 Какие операции отношения вы знаете?

5 В чем особенность использования операторов if и switch?

## 4 Лабораторная работа № 4. Отладка программ

**Цель работы**: получение навыков отладки программного кода.

Теоретические сведения и содержание работы приведены в [1, с. 30–31].

Индивидуальные задания: задание 1 [1, таблица 2.2, с. 20]; задание 2 [1, таблица 2.3, с. 22].

### *4.1 Теоретические сведения*

Для поиска логических ошибок используется встроенный отладчик.

Для пошагового выполнения программы необходимо нажимать клавишу F10. При каждом нажатии выполняется текущая строка. Если необходимо пошагово проверить текст вызываемой функции, то следует нажать F11. Для досрочного выхода из функции нажать Shift + F11. Если необходимо начать отладку с определенного места программы, то надо установить курсор в соответствующую строку программы и нажать Ctrl + F10.

Другим способом отладки является установка точек прерывания программы. Для этого надо поместить курсор в нужную строку и нажать F9. Точка прерывания обозначается красным кружком на специальном поле, расположенном слева от окна текста программы. Для удаления точки прерывания следует в необходимой строке повторно нажать F9. Количество точек прерывания в программе может быть любым.

Для выполнения программы до точки прерывания необходимо нажать F5. Для продолжения отладки нажимается клавиша F5 (для выполнения программы до следующей точки прерывания) или используются клавиши для пошаговой отладки.

Желтая стрелка на поле слева от окна текста программы указывает на строку, которая будет выполнена на следующем шаге отладки.

Для контроля за значениями переменных удобно использовать следующий способ: подвести указатель мыши к интересующей переменной и задержать его на несколько секунд. На экране рядом с именем переменной появится окно, содержащее текущее значение этой переменной. Кроме этого, значения переменных будут отображаться в окнах, расположенных снизу. В левом нижнем окне отображаются значения последних использованных программой переменных. В правом нижнем окне (Watch) можно задать имена переменных, значения которых необходимо контролировать.

### *4.2 Индивидуальные задания*

1 Расположить в порядке возрастания значения, полученные в результате вычисления выражений a*ln(b), tg(a) + b/a, sin(a)*b с указанием формул, по которым производились вычисления.

2 Элементы окружности пронумерованы таким образом: 1 – радиус (R), 2 – диаметр (D), 3 – длина (L), 4 – площадь круга (S). По номеру элемента и его значению вычислить значения остальных элементов.

3 Даны произвольные числа a и b. Поменять их значения так, чтобы стало a <= b <= c.

### *Контрольные вопросы*

1 Какие виды ошибок могут появляться при разработке программного кода?

2 Для чего предназначен отладчик Visual Studio?

3 Перечислите основные команды отладки кода, доступные в Visual Studio/

# 5 Лабораторная работа № 5. Циклические программы

**Цель работы**: получение навыков программирования циклических алгоритмов.

## *5.1 Теоретические сведения*

Оператор цикла for. Общий вид оператора

```
for (инициализирующее выражение; условие; инкрементирующее выражение)
{
 тело цикла
}
```

Инициализирующее выражение выполняется только один раз в начале выполнения цикла и, как правило, инициализирует счетчик цикла.

Условие содержит операцию отношения, которая выполняется в начале каждого цикла. Если условие равно 1 (true), то цикл повторяется, иначе выполняется следующий за телом цикла оператор.

Инкрементирующее выражение, как правило, предназначено для изменения значения счетчика цикла. Модификация счетчика происходит после каждого выполнения тела цикла.

Оператор цикла while. Оператор цикла с предусловием. Общий вид оператора

```
while (условие)
{
        тело цикла
}
```

Организует повторение операторов тела цикла до тех пор, пока условие истинно.

Оператор цикла do. Оператор цикла с постусловием. Общий вид оператора

```
do
{
        тело  цикла
} while (условие);
```

Организует повторение операторов тела цикла до тех пор, пока условие истинно.

***Пример*** – Из n целых чисел, введенных с клавиатуры, определить максимальную последовательность четных чисел.

```
int main ()
{
```

```
    setlocale (LC_ALL, "Russian");
    int n, x;
    int cnt=0;
    int k=0;
    cout<<"Введите последовательность из ";
    cin>>n;
    cout<<"чисел\n";
    for (int i=1; i<=n; i++)
    {
        cin>>x;
        cnt=0;
        while (x%2==0)
        { cnt++;
        if (i==n) break;
        cin>>x;
        i++;
     }
     if (cnt>k)
        k=cnt;
     cout<<"Количество четных равно "<<k<<endl;
}
```

### *5.2 Индивидуальные задания*

1 Даны начальное значение $a_0$ = 5 и рекуррентная формула $a_i = a_{i-1} + 3i/2$. Найти номер первого элемента, превысившего введенное с клавиатуры число.

2 Даны начальное значение $a_0$ = 1 и рекуррентная формула $a_i = a_{i-1}/2i$. Найти наименьший номер элемента последовательности, для которого выполняется условие $|a_i - a_{i-1}| < \varepsilon$, введенное с клавиатуры. Вывести на экран этот номер и все элементы $a_i$.

3 Даны начальное значение $a_0$ = 2 и рекуррентная формула $a_i = 2i + 1/a_{i-1}$. Найти номер первого элемента, превысившего введенное с клавиатуры число.

### *Контрольные вопросы*

1 Каким образом цикл while может имитировать цикл for?
2 Каким образом цикл while может имитировать цикл do-while?
3 В каких случаях используются операторы break, continue, exit?
4 Почему в языке С++ нет необходимости использовать оператор goto?

# 6 Лабораторная работа № 6. Одномерные массивы

**Цель работы**: изучение способов объявления, инициализации, организации ввода-вывода и обработки одномерных массивов; получение навыков алгоритмизации и программирования прикладных задач с применением одномерных массивов.

## *6.1 Теоретические сведения*

В программе одномерный массив объявляется следующим образом:

тип имя массива [размер];

Пример декларации массива:

int mas[4];

Индексы в массиве начинаются с 0 (т. е. массив, приведенный в примере, будет содержать следующие элементы: mas[0], mas[1], mas[2] и mas[3]). Выход индекса за пределы массива не проверяется.

***Пример*** – Удалить из одномерного массива все отрицательные элементы.

```
for (int i=0; i<n; ++i)
      if (a[i]<0)
      {
            for (j=i+1; j<n; j++)
                  a[j-1]=a[j];
            n--;   i--;
      }
```

## *6.2 Индивидуальные задания*

1 Определить, является ли данный целочисленный массив упорядоченным по убыванию. Если да, то заменить его минимальный элемент на сумму предшествующих элементов, если нет, то заменить его максимальный элемент на сумму последующих элементов.

2 Создать массив из номеров нулевых элементов исходного массива и сжать его, выбросив все нулевые элементы.

3 В массиве вещественных чисел определить, сколько раз меняется знак, и вывести номера позиций, в которых происходит смена знака.

## *Контрольные вопросы*

1 Объясните, что такое массив.
2 Назовите способы объявления одномерного массива.

3 Назовите способы инициализации одномерного массива.

4 Какие значения принимают элементы массива при отсутствии в его объявлении инициализатора?

# 7 Лабораторная работа № 7. Обработка символьной информации

**Цель работы**: получение навыков написания программ для обработки строк и символьной информации.

## *7.1 Теоретические сведения*

Переменную строкового типа можно описать несколькими способами.

1 Описать как массив символов:

а) char имя_массива [n]; где n – количество символов в строке, включая завершающий нульсимвол. Например, char stroka[10];

б) char имя_массива [ ]. Например, char s[ ];

В описании а) для переменной stroka будет выделено 10 байт памяти. В описании б) память выделена не будет. Это описание означает, что переменная s – это переменная строкового типа, длина которой не определена.

2 Описать с использованием указателя на тип char, например, char *stroka. В данном описании память для переменной stroka не выделяется. Такое описание означает, что переменная stroka может содержать адрес ячейки памяти первого символа строки. Выделить память для переменной stroka можно с помощью функции new:

```
char *stroka=new char[20];
```

где 20 – объем памяти, выделенной для строки.

***Пример*** – Ввод строки, содержащей пробелы.

```
int main ()
{
      const int MAX_LEN = 80; // максимальная длина строки
      char str[MAX_LEN];
      cout << "Input string : ";
      cin.get (str,MAX_LEN);
      cout << "Inputed string :" << str << endl;
}
```

## *Контрольные вопросы*

1 Что представляют собой строки?

2 Каким образом строки описываются и определяются?

3 Какие функции используются для ввода строки и чем они отличаются друг от друга?

4 Какие функции используются для вывода строки?

# 8 Лабораторная работа № 8. Многомерные массивы

**Цель работы**: изучение способов объявления, инициализации, организации ввода-вывода и обработки двумерных массивов; получение навыков алгоритмизации и программирования прикладных задач с применением двумерных массивов.

## *8.1 Теоретические сведения*

В многомерном массиве каждый дополнительный индекс обеспечивает дополнительное средство доступа к конкретному элементу, или дополнительное измерение. Наиболее распространенным видом многомерного массива являются двумерные массивы, или матрицы.

Общий синтаксис для объявления двумерных и трехмерных массивов

```
тип_элемента имя_массива [N_1] [N_2];
тип_элемента имя_массива [N_1] [N_2] [N_3];
```

Как и в одномерных массивах, каждый индекс имеет нижнюю границу, равную 0, а число элементов для каждого уровня индекса определяется при объявлении многомерного массива.

Пример объявления многомерных массивов

```
double matrix A [100][10];
char table [41][22][3];
int index [7][12];
```

Большинство компиляторов хранит элементы многомерного массива друг за другом, т. е. как длинный одномерный массив. Исполняемый модуль вычисляет, где расположен искомый элемент в этом массиве.

Однако заполнение и обработку многомерных массивов чаще всего производят, пользуясь вложенными циклами.

Язык C++ дает возможность инициализировать многомерный массив способом, аналогичным инициализации одномерных массивов. Для этого нужно использовать список значений, расположенных в той же последовательности, в которой элементы многомерного массива хранятся в памяти.

## *8.2 Индивидуальные задания*

1 Дана целочисленная матрица размером 6 × 9, содержащая как положительные, так и отрицательные элементы. Сформировать одномерные массивы, состоящие из средних арифметических значений положительных элементов каждого

столбца матрицы и средних арифметических значений нечетных элементов каждой строки матрицы.

2 Дана целочисленная матрица размером 7 × 10. Поменять местами максимальное и минимальное значения матрицы. Найти суммы элементов тех столбцов, где находятся эти значения.

3 Дана целочисленная матрица размером 5 × 8. Найти сумму четных элементов каждого столбца правой половины матрицы и среднее арифметическое значение нечетных элементов каждого столбца левой половины матрицы. Сформировать соответствующие одномерные массивы.

***Контрольные вопросы***

1 Дайте определение двумерного массива.
2 Назовите способы объявления двумерного массива.
3 Как осуществляется обращение к элементам двумерного массива?
4 Назовите способы инициализации двумерного массива.

# 9 Лабораторная работа № 9. Указатели как параметры и результаты функций

**Цель работы**: получение навыков написания программ с использованием указателей.

## *9.1 Теоретические сведения*

Функция – это последовательность операторов, оформленная таким образом, что ее можно вызвать по имени из любого места программы. Функция описывается следующим образом:

```
тип возвращаемого значения имя функции (список параметров)
      {
       тело функции
}
```

Первая строка описания называется заголовком функции. Тип возвращаемого значения может быть любым, кроме массива или функции. Допустимо не возвращать никакого значения (тип void).

В C++ не допускается вложение функций друг в друга.

Выход из функции осуществляется следующими способами.

1 Если нет необходимости возвращать вычисленное значение, то выход осуществляется по достижении закрывающей скобки или при выполнении оператора return.

2 Если необходимо вернуть определенное значение, то выход осуществляется оператором

return выражение;

Передача параметров в функцию. При работе важно соблюдать следующее правило: при объявлении и вызове функции параметры должны соответствовать по количеству, порядку следования и типам. Функция может не иметь параметров, в этом случае после имени функции обязательно ставятся круглые скобки. Существует три основных способа передачи параметров: передача по значению, ссылке или указателю.

Передача параметров по значению. В момент обращения к функции в памяти создаются временные переменные с именами, указанными в списке параметров. Во временные переменные копируются значения фактических параметров.

Передача параметров по ссылке. При передаче параметров по ссылке передается не значение соответствующей переменной, а ее адрес. Для указания на данный способ передачи после имени параметра ставится символ «&».

Передача параметров по указателю. В отличие от передачи по ссылке адрес переменной передается в функцию не с использованием операции разадресации (&), а операцией косвенной адресации (*).

***Пример*** – Вывести на экран таблицу значений функции $Y(x) = \sin x$ и ее разложения в ряд $S(x) = x - \frac{x^3}{3!} + ... + (-1)^n \frac{x^{2n+1}}{(2n+1)!}$ с точностью ε = 0,001. Вывести число итераций, необходимое для достижения заданной точности.

```
#include <iostream.h>
#include <math.h>
#include <iomanip.h>
typedef double (*uf)(double, double, int &);

void tabl(double, double, double, double, uf);
double y(double, double, int &);
double s(double, double, int &);

int main()
{
      cout << setw(8) <<"x"<< setw(15) <<"y(x)"<< setw(10) << "k" << endl;
      tabl(0.1,0.8,0.1,0.001,y);
      cout << endl;
      cout << setw(8) <<"x"<< setw(15) <<"s(x)"<< setw(10) << "k" <<endl ;
      tabl(0.1,0.8,0.1,0.001,s);
      return 0;
}

void tabl(double a, double b, double h, double eps, uf fun)
{
      int k=0;
```

```
        double sum;
        for (double x=a; x<b+h/2; x+=h)
        {
                sum=fun(x,eps,k);
                cout << setw(8) << x << setw(15) << sum << setw(10) << k << endl;
        }
}

double y(double x, double eps, int &k)
{
        return sin(x);
}

double s(double x, double eps, int &k)
{
        double a,c,sum;
        sum=a=c=x;
        k=1;
        while (fabs(c)>eps)
        {
                c = pow(x,2)/(2*k*(2*k+1));
                a *= -c;
                sum += a;
                k++;
        }
        return sum;
}
```

### *9.2 Индивидуальное задание*

Вывести на экран таблицу значений функции $Y(x)$ и ее разложения в ряд $S(x)$ с точностью ε (таблица 9.1). Вывести число итераций, необходимое для достижения заданной точности. Вычисление $S(x)$ и $Y(x)$ оформить в виде функций.

Таблица 9.1

| Номер варианта | $a$ | $b$ | $S(x)$ | ε | $Y(x)$ |
|---|---|---|---|---|---|
| 1 | –0,9 | 0,9 | $x+\frac{x^3}{3}+...+\frac{x^{2k+1}}{2k+1}$ | $10^{-4}$ | $\frac{1}{2}\ln\frac{1+x}{1-x}$ |
| 2 | 0,1 | 0,9 | $1-\frac{x^2}{3!}+...+(-1)^k\frac{x^{2k}}{(2k+1)!}$ | $10^{-5}$ | $\frac{\sin x}{x}$ |
| 3 | –0,9 | 0,9 | $1+\frac{x}{3}+\sum_{k=2}^{\infty}(-1)^{k-1}\frac{1\cdot2\cdot5\cdot8\cdot...(3k-4)}{3^k k!}x^k$ | $10^{-3}$ | $\sqrt[3]{1+x}$ |

***Контрольные вопросы***

1 Что хранится в переменной-указателе?
2 Указателем на какой элемент массива является имя массива?
3 Обнуляется ли динамическая память при выделении?
4 С помощью какой операции производится освобождение памяти, выделенной посредствам new()?

# 10 Лабораторная работа № 10. Массивы и ссылки при работе с функциями

**Цель работы**: получение навыков передачи массивов в функции.

## *10.1 Теоретические сведения*

Рассмотрим передачу массивов, точнее их адресов, в качестве аргументов функции. Для передачи одномерного массива достаточно объявить функцию, которая примет в качестве параметра массив значений типа double и вычислит сумму элементов этого массива. Прототип этой функции можно записать таким образом:

```
double SumOfData (double data[max]); // при константном
                                     // значении max
```

Еще лучше объявить

```
double SumOfData (double data[ ], int n);
```

При объявлении функции SumOfData() ей можно передать массив аргументов вместе с целым числом n, сообщающим, сколько элементов содержится в передаваемом массиве.

Сама функция SumOfData() может содержать, например, обычный цикл while, суммирующий элементы массива, а функция return возвращает результат:

```
double SumOfData (double data[ ], int n)
{
    double sum=0;
    while (n>0)
    sum += data [--n];
    return sum;
}
```

При передаче двумерных массивов необходимо сообщить компилятору количество столбцов для вычисления адреса элементов в начале каждой строки.

Пусть ROWS – количество строк, COLS – количество столбцов. Объявим функцию с массивом в качестве параметра:

```
void AnyFn (int data[ROWS][COLS]);
```

или

```
void AnyFn (int data[ ][COLS]);
```

Можно передать параметр, определяющий количество строк:

```
void AnyFn (int data[ ][COLS], int numRows);
```

где numRows сообщает функции число строк в массиве data.

***Пример*** – Передача функции многомерных массивов.

```
const int COLS=8;
void FillArray (int data[ ][COLS], int numRows);
void DisplayTable (int data[ ][COLS], int numRows);
int data1[7][COLS];
int data2[4][COLS];

int main ( )
{
      time_t t;
      // инициализация генератора случайных чисел
      srand (time (&t));
      FillArray (data1, 7);
      DisplayTable (data1, 7);
      FillArray (data2,4);
      DisplayTable (data2,4);
      return 0;
}

void FillArray (int data[ ][COLS], int numRows)
{
      int r, c;
      for (r=0; r<numRows; r++)
         for (c=0; c<COLS; c++)
               data[r][c]= rand()%100;
}

void DisplayTable (int data[ ][COLS], int numRows)
{
      int r, c;
```

```
    for (r=0; r<numRows; r++)
    {
       cout<<endl;
       for (c=0; c<COLS; c++)
              cout<<setw (5)<<data[r][c];
    }
    cout<<endl;
}
```

### *10.2 Индивидуальные задания*

1 Объединить два упорядоченных по возрастанию целочисленных массива (M1[10] и M2[5]) в один, не нарушая упорядоченности.

2 Заменить нечетные элементы исходного 20-элементного массива, состоящего из целых чисел, нулевыми значениями.

3 Создать массив из 20 символьных значений. Сформировать из его значений три других массива, состоящих: из цифр, из букв и из символов, не являющихся ни буквами, ни цифрами.

### *Контрольные вопросы*

1 Перечислите способы передачи одномерного массива в функцию в качестве входного параметра.

2 Перечислите способы передачи двумерного массива в функцию в качестве входного параметра.

3 Как возвратить результат работы функции в виде массива?

## 11 Лабораторная работа № 11. Представление информации в виде структуры

**Цель работы**: получение навыков разработки программ с использованием структур.

### *11.1 Теоретические сведения*

Для определения структурного типа используется ключевое слово struct:

```
struct имя_структурного_типа
{ описание поля 1 ;
. . .
описание поля n ;
} [одна или более переменных];
```

Поле описывается как переменная стандартного типа или типа, определенного пользователем.

***Пример*** – Структура с информацией о компакт-дисках (CD):

```
struct cd_info
{
      char titlel[25]; // название CD
      char regiss [20]; // режиссер
      int num_films; // число фильмов
      float price; // стоимость CD
      char date_bought[8]; // дата покупки
} col1,col2,col3;
```

Переменные структурного типа можно объявлять так же, как и переменные стандартных типов.

При определении структурной переменной язык С++ резервирует для нее место в памяти. Если же был описан только структурный тип, а ни одной переменной данного типа определено не было, то место в памяти не выделяется.

### *11.2 Индивидуальные задания*

1 Сформировать массив, содержащий сведения о количестве изделий, собранных рабочими цеха за неделю. Структурный тип содержит поля: фамилия сборщика; количество изделий, собираемых им ежедневно в течение шестидневной недели, т. е. раздельно в понедельник, вторник и т. д. Написать программу, выдающую на печать фамилию рабочего и общее количество деталей, собранных им за неделю.

2 Сформировать массив, содержащий сведения о количестве изделий категорий A, B, C, собранных рабочим за месяц. Структурный тип содержит поля: фамилия сборщика, наименование цеха, количество изделий по категориям, собранных рабочим за месяц. Считая заданными значения расценок SA, SB, SC за выполненную работу по сборке единицы изделия категорий A, B, C, вывести на экран общее количество изделий категорий A, B, C, собранных рабочим цеха.

3 Сформировать массив, содержащий сведения о телефонах абонентов. Структурный тип содержит поля: фамилия абонента, место жительства (название улицы, номер дома), год установки телефона. Написать программу, выдающую информацию о количестве установленных телефонов с XXXX (ввести с клавиатуры) года.

### *Контрольные вопросы*

1 Какова область применения структур?
2 Каким образом определяется структура?
3 Как определяются переменные типа структура?
4 Как осуществляется доступ к структурным членам?

# 12 Лабораторная работа № 12. Программирование рекурсивных алгоритмов

**Цель работы**: получение навыков разработки программ с использованием рекурсивных функций.

## *12.1 Теоретические сведения*

Рекурсия – это такой способ организации вычислительного процесса, при котором процедура или функция в ходе выполнения составляющих ее операторов обращается сама к себе.

В рекурсивной процедуре или функции обязательно должно содержаться условие окончания рекурсии.

При выполнении рекурсии, когда подпрограмма доходит до рекурсивного вызова, процесс приостанавливается и новый процесс запускается с начала, но уже на новом уровне. Прерванный же процесс запоминается. Новый процесс тоже может быть приостановлен и т. д. Образуется последовательность прерванных процессов.

Последовательность рекурсивных обращений должна обязательно выходить на определенное значение.

При каждом очередном входе в подпрограмму ее локальные параметры и адреса возврата размещаются в специальной области памяти – стеке. После выхода на определенной значение выполняется обратный ход – цепочка вычислений по обратному маршруту, сохраненному с стеке.

Рассмотрим это на примере функции вычисления факториала. Хорошо известно, что 0! = 1, 1! = 1. А как вычислить величину n! для большого n? Если бы мы могли вычислить величину (n – 1)!, то тогда легко вычислим n!, поскольку n! = n * (n – 1)!. Но как вычислить (n – 1)!? Если бы вычислили (n – 2)!, то сможем вычислить и (n – 1)! = (n – 1) * (n – 2)!. А как вычислить (n – 2)!? Если бы ... . В конце концов, мы дойдем величины 0!, которая равна 1. Таким образом, для вычисления факториала можем использовать значение факториала для меньшего числа.

Программа соответствующего алгоритма имеет вид

```
int factorial (int n)
{
      if (n == 0)
      {
         return 1;
      }
      else
      {
```

```
        return n * factorial(n – 1);
    }
}
```

### *12.2 Индивидуальные задания*

1 Найти сумму цифр заданного натурального числа.

2 Написать программу вычисления целой степени любого вещественного числа.

3 Составить программу для нахождения числа, которое образуется из данного натурального числа при записи его цифр в обратном порядке.

### *Контрольные вопросы*

1 Дайте определение рекурсивной функции.

2 Опишите синтаксис вызова рекурсивной функции на выполнение.

3 Что является элементами класса?

4 Как осуществляется защита от бесконечного вызова рекурсивной функции?

## 13 Лабораторная работа № 13. Динамические структуры данных. Списки

**Цель работы**: получение навыков разработки программ с использованием динамической структуры данных список.

### *13.1 Теоретические сведения*

Как и обычные структуры, динамические структуры состоят из полей, или членов структур. Но кроме информационных полей, они обязательно содержат поля-указатели на свой собственный тип структуры. Поэтому динамические структуры часто называются самоссылочными структурами. Среди динамических структур наиболее часто встречаются очереди (списки), стеки, деревья, двунаправленные списки.

Самоссылочную структуру можно объявить таким образом:

```
struct node {
    int info; // здесь может находиться любое
                        // информационное поле, и их
                        // может быть больше, чем одно
    struct node *next; // указатель на структуру
                       // типа node
}
```

Структура объявленного типа имеет информационное поле целого типа и указатель на такую же структуру, с помощью которого структуры объединяются в списки. Списки очень удобны для хранения различной информации.

Важно, что указатель содержит адрес структуры (иногда называемую узлом) целиком, а не указывает на отдельные компоненты, находящиеся внутри нее.

Формирование очереди. Для создания списка нужно объявить структуру аналогично структуре node:

```
struct node {
        int info;
        struct node *next;
};
```

и указатель на объявленный структурный шаблон:

```
typedef node *NodePtr; //указатель на тип node
```

После этого объявляют указатель на начало списка, который иногда называют заголовком списка. Назовем указатель head и объявим его:

```
NodePtr head = NULL;
```

Нулевой указатель на начало списка является признаком того, что список пустой. Для начала формирования списка, т. е. для размещения первого элемента в списке, следует выделить память под этот элемент, предварительно убедившись, что список пустой. Это достигается путем выполнения операторов:

```
if (head == NULL)
{
        head = new node;
        head->info = 1; // или какому-то другому значению
        head->next = NULL;
}
```

Оператор p->info = 1; эквивалентен оператору (*p).info = 1;. Первый из них называется операцией косвенного выбора и объединяет в себе действия, связанные с разыменованием и выбором поля структуры. Второй оператор осуществляет прямой выбор. Не следует забывать круглые скобки вокруг *p, так как приоритет операции разыменования ниже, чем приоритет операции обращения к полю структуры.

Для выделения памяти можно пользоваться любым способом резервирования памяти.

Список из одного такого элемента имеет вид, представленный на рисунке 13.1.

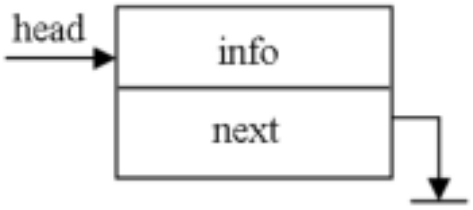



Рисунок 13.1 – Список из одного элемента

Но список редко состоит из одного элемента. Поэтому следующим шагом является добавление нового элемента в список. Для этого необходимо объявить указатель на текущий элемент p и выполнить следующие действия.

```
NodePtr p; //указатель на текущий элемент
NodePtr tail; //указатель на "хвост" очереди
if (head == NULL)
{
      head = new node;
      head->info = 1; // или какому-то другому значению
      head->next = NULL;
}
p = new node;
p->info = 2;
p->next = head->next; // в данном случае – NULL
head->next = p;
tail = p;
```

После вставки элемента список приобретает вид, показанный на рисунке 13.2.

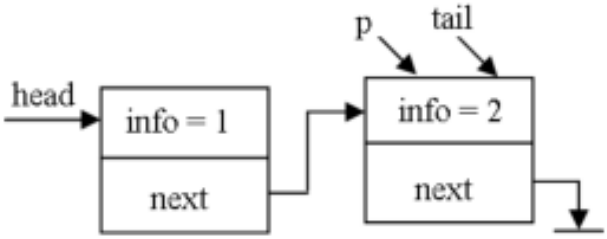



Рисунок 13.2 – Список из двух элементов

После вставки аналогичным образом ещё нескольких элементов, список приобретает вид, показанный на рисунке 13.3. Если при этом известно количество элементов в очереди, используют оператор цикла с параметром:

```
NodePtr p; // указатель на текущий элемент
NodePtr tail; // указатель на "хвост" очереди
int N = 10; // количество элементов в очереди
```

```
int cnt = 1; // счетчик элементов в очереди
head = NULL;
. . .
if (head == NULL)
{
        head = new node;
        head->info = cnt++; // или какому-то другому значению
        head->next = NULL;
        tail = head;
}
for (int i = 2; i<=N; i++)
{
        p = new node;
        p->info = cnt++;
        tail->next = p; // в данном случае – NULL
        p->next = NULL;
        tail = p;
}
```

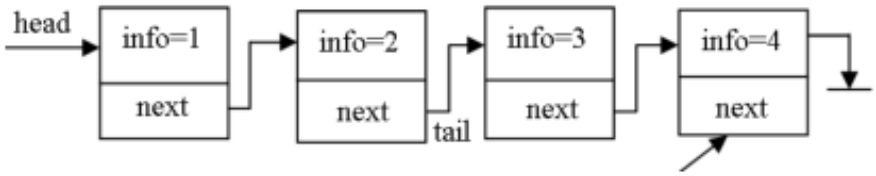



Рисунок 13.3 – Список из нескольких элементов

Создали список или очередь (queue, FIFO, First Input-First Output, Первым вошел – Первым вышел), поэтому заголовочный элемент (head) всегда указывает на первый элемент списка.

### *13.2 Индивидуальные задания*

1 Используя динамическую структуру очередь, описать функцию, которая считает сумму элементов списка, стоящих за каждым вхождением элемента E, значение которого введено с клавиатуры.

2 Используя динамическую структуру очередь, описать функцию, которая находит среднее арифметическое значение всех элементов сформированного непустого списка.

3 В составе программы описать функцию, которая в списке, состоящем из символьных элементов, определяет количество буквенных символов.

### *Контрольные вопросы*

1 Что такое динамическая структура данных очередь и с какой целью она используется?

2 Как оформляется очередь?

3 Какие преимущества программы обеспечиваются при использовании очереди?

# 14 Лабораторная работа № 14. Структура данных стек

**Цель работы**: получение навыков разработки программ с использованием структуры данных стек.

## *14.1 Теоретические сведения*

Стеком называется структура данных, организованная по принципу: «Последним вошел – первым вышел» (LIFO, Last Input – First Output).

Аналог стека – бусинки, которые нанизаны на нитку. Последнюю из нанизанных бусинок снять легко, а чтобы добраться до самой первой, нужно снять все. Как и очередь, стек организован на основе самоссылочных структур (рисунок 14.1). Поэтому для формирования стека объявим такую же структуру, как и для формирования очереди:

```
struct node {
        int info;
        struct node *next;
};

typedef node *NodePtr; //указатель на тип Node
```

Первым в стек помещен элемент с информационным полем, равным 1, вторым – 2 и т. д. Как видно из рисунка 14.1, следуя по указателям next от элемента head, доберемся до элемента 1 в последнюю очередь.

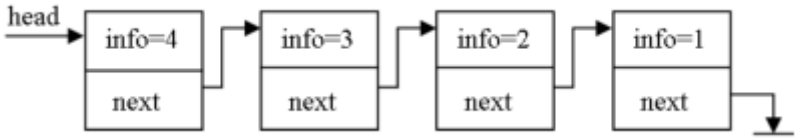



Рисунок 14.1 – Стек

Формирование стека можно осуществить с помощью следующих операторов:

```
NodePtr head = NULL;
if (head == NULL)
{
        head = new node;
        head->info = random (100) – 50;
```

```
        head->next = NULL;
}
else {
        p = new node;
        p->info = random (100) – 50;
        p->next = head;
        head = p;
}
```

Выбор оператора цикла при создании стека происходит в зависимости от того, известно ли количество элементов в стеке или известен признак завершения его формирования.

***Пример*** – Программа формирования стека из 10 элементов и вывода его на экран.

```
int main ( )
{
     struct node { int info;
          node *next;
     };
     typedef node *NodePtr;// указатель на тип node
     NodePtr head = NULL;
     NodePtr p; // указатель на текущий элемент
     int N = 10; // количество элементов в стеке
     int cnt = 1; // счетчик элементов в стеке
     if (head == NULL)
     {
          head = new node;
          head->info = cnt++; //или так: rand()%100-50;
          head->next = NULL;
     }
     for (int i = 2; i <= N; i++)
     {
          p = new node;
          p->info = cnt++; //или rand()%100-50;
          p->next = head;
          head = p;
     }
     // Вывод стека на экран
     p = head;
     for (int i = 1; i <= N; i++)
     {
          cout << p->info << ' ';
          p = p->next;
```

```
    }
    cout <<endl;
}
```

### *14.2 Индивидуальные задания*

1 Используя динамическую структуру стек, описать функцию, которая считает сумму элементов списка, стоящих перед каждым вхождением элемента E, значение которого введено с клавиатуры.

2 Используя динамическую структуру стек, описать функцию, которая в списке, состоящем из символьных элементов, определяет количество цифровых символов.

3 Используя динамическую структуру стек, описать функцию, которая заменяет в списке все вхождения элемента E1, значение которого введено с клавиатуры, на элемент E2, значение которого также введено с клавиатуры.

### *Контрольные вопросы*

1 Что такое динамическая структура данных стек и с какой целью они используются?

2 Как оформляется стек?

3 Какие преимущества программы обеспечиваются при использовании стека?

## 15 Лабораторная работа № 15. Классы и объекты

**Цель работы**: получение навыков разработки программ с использованием классов.

### *15.1 Теоретические сведения*

Класс – это расширение понятия структуры языка C++. Он позволяет создавать типы и определять функции, которые задают поведение типа. Каждый представитель класса называется объектом.

Определение класса идентично определению структуры в языке C++, но имеет и отличия:

– обычно содержит одну или несколько спецификаций доступа (public, protected, private);

– вместо ключевого слова struct используется слово class ;

– обычно включает в себя функции (функции-элементы или методы) наряду с данными-элементами;

– обычно в нем имеются некоторые специальные функции, такие как конструктор (функция с тем же именем, что и сам класс) и деструктор (функция, именем которой является имя класса с префиксом тильда (~)).

***Пример*** – Определение класса.

```
class str
{
      char *s;                    //элемент-данное
      public:                      //спецификатор открытого доступа
      str (char *word);            //функция-элемент: конструктор
      ~str();                      //функция-элемент: деструктор
      void write();                //функция-элемент: метод печати
};
```

В языке C++ можно ограничить видимость данных и функций класса при помощи меток public, protected, private. Метка-спецификатор доступа применяется ко всем элементам класса, следующим за ней, пока не встретится другая метка или кончится определение класса.

Метка-спецификатор public (открытый) используется тогда, когда данные-элементы и функции-элементы класса должны быть доступны для функций-элементов и других функций программы, в которой имеется представитель класса.

Метка-спецификатор protected (защищенный) используется в том случае, когда элементы данных и функции-элементы должны быть доступны для функций-элементов данного класса и классов производных от него.

Метка-спецификатор private (закрытый) используется, если данные-элементы и функции-элементы должны быть доступны только для функций-элементов данного класса.

В классе элементы по умолчанию являются закрытыми.

Элементы класса делятся на две основные категории:

1) данные, называемые данными-элементами;

2) код, называемый функциями-элементами, или методами.

Данные-элементы классов языка C++ идентичны элементам структур языка C++ с некоторыми дополнениями:

– данными-элементами могут быть перечислимые типы, битовые поля или представители ранее объявленного класса. Также допускается вложенное объявление перечислимого типа данных и создание псевдонимов с помощью typedef;

– данное-элемент класса может быть указателем или ссылкой на представитель этого класса.

Функция-элемент класса является функцией, объявленной (описанной) внутри определения класса. Тело функции может также определяться внутри определения класса, в этом случае функция называется встроенной (inline) функцией-элементом. Когда тело функции определяется вне тела класса, перед именем функции ставится префикс из имени класса и операции разрешения видимости (::).

***Пример.***

```
class str
{
     char *s; // указатель на строку
     public:
     str (char *word) // встроенный конструктор
     {
           s=new char[strlen (word)+1];
           strcpy (s, word);
     };
     ~str()
     {
          delete [ ]s;
     }; // встроенный деструктор
     void write(); // объявление функции-элемента
};

void str::write() // определение функции-элемента
{
          cout<<s;
};
```

Доступ к данным-элементам класса. Функции-элементы класса находятся в области действия класса, в котором они определены, т. е. они могут обращаться к любому элементу класса, используя просто имя переменной. Обычные функции или функции-элементы другого класса могут получить доступ к данным элементам с помощью операции . или −>, применяемых к представителю или указателю на представитель класса.

***Пример.***

```
class coord
{
          public: int x, y; // координаты x и y
};

void main()
{
      coord org; // представитель класса координат
      coord *orgptr = &org; // указатель на представитель
      // класса
      org.x = 0; // задание значения координаты x
      orgptr−>y = 0; // задание значения координаты y
}
```

Каждая нестатическая (не имеющая спецификатора static) функция-элемент класса имеет доступ к объекту, для которого вызвана, через ключевое слово this. Указатель this является указателем на тип_класса *.

Конструктор инициализирует представитель класса (объект) и является функцией-элементом с тем же именем, что и класс. Конструктор вызывается компилятором всегда, когда создается представитель класса. Объект считается созданным в тот момент, когда завершил работу конструктор объекта.

Деструктор является дополнением конструктора. Он имеет то же имя, что и класс, но с префиксом тильда (~). Он вызывается всякий раз, когда уничтожается представитель класса. Объект считается уничтоженным, когда завершил работу деструктор объекта.

### *15.2 Индивидуальные задания*

1 Создать класс «Двигатель автомобиля», включающий данные-элементы: марка а/м, объем топливного бака, расход бензина на 100 км.

Функции-элементы:

– создание и инициализация (конструктор);

– заправка а/м (количество бензина – в аргументе);

– расход бензина (пройденный путь – в аргументе);

– выдача сообщения о том, сколько километров можно проехать на оставшемся бензине;

– деструктор.

2 Создать класс «Мобильный телефон», включающий данные-элементы: номер, имя владельца, количество денег на счете.

Функции-элементы:

– создание и инициализация (конструктор);

– пополнение счета (сумма – в аргументе);

– оплата разговоров (тариф и время – в аргументе);

– выдача сообщения об остатке средств на счете;

– деструктор.

3 Создать класс «Платная автостоянка», включающий данные-элементы: название, место расположения, количество мест, тариф.

Функции-элементы:

– создание и инициализация (конструктор);

– количество уехавших машин (в аргументе);

– количество машин, желающих встать на стоянку (в аргументе);

– выдача сообщения о количестве свободных мест;

– деструктор.

### *Контрольные вопросы*

1 Дайте определение класса и объекта.

2 Какие члены могут входить в состав класса?

3 Как создаются объекты класса?

# 16 Лабораторная работа № 16. Наследование

**Цель работы**: получение навыков разработки программ с использованием наследования.

## *16.1 Теоретические сведения*

Язык C++ позволяет классу наследовать данные-элементы и функции-элементы одного или нескольких других классов. Новый класс называют производным классом. Класс, элементы которого наследуются производным классом, называется базовым классом. В свою очередь производный класс может служить базовым для другого класса. Наследование дает возможность заключить некоторое общее или схожее поведение различных объектов в одном базовом классе.

Наследование позволяет также изменить поведение существующего класса. Производный класс может переопределить некоторые функции элементы базового, наследуя, тем не менее, основной объем свойств и атрибутов базового класса. Общий вид наследования

```
class Base
{
// …..
};

class Derived: <ключ доступа> Base
{
// …
};
```

Ключ доступа может быть private, protected, public. Если ключ не указан, то по умолчанию он принимается private.

Наследование позволяет рассматривать целые иерархии классов и работать со всеми элементами одинаково, приводя их к базовому. Правила приведения следующие:

– наследуемый класс всегда можно привести к базовому;

– базовый класс можно привести к наследуемому, только если в действительности это объект наследуемого класса.

Конструкторы не наследуются. Если конструктор базового класса требует спецификации одного или нескольких параметров, конструктор производного класса должен вызывать базовый конструктор, используя список инициализации элементов.

***Пример.***

```
#include <string.h>
class Base
{
```

```
public:
        Base (int, float);
};

class Derived: Base
{
public:
        Derived (char* lst, float amt);
};

Derived:: Derived (char* lst, float amt) : Base (strlen (lst),amt)
{ }
```

### *16.2 Индивидуальные задания*

1 Разработать программу с использованием наследования классов, реализующую классы: Человек (Имя, дата рождения) –> Школьник(Номер школы) –> Студент (Название вуза, специальность). Вывести на экран характеристики объектов.

2 Разработать программу с использованием наследования классов, реализующую классы: Колесо (Радиус) –> Садовая тележка (Количество колес, грузоподъемность) –> Грузовик (Марка, мощность). Вывести на экран характеристики объектов.

3 Разработать программу с использованием наследования классов, реализующую классы: Точка (Координаты) –> Отрезок: координаты начала наследуются из точки (Координаты конца, длина) –> Прямоугольный треугольник: один из катетов равен длине отрезка, другой – половине длины отрезка (Площадь). Вывести на экран характеристики объектов.

### *Контрольные вопросы*

1 Какие ключи доступа используются при наследовании?
2 Наследуются ли конструкторы?
3 Наследуются ли деструкторы?

# 17 Лабораторная работа № 17. Полиморфизм

**Цель работы**: получение навыков разработки программ с использованием полиморфизма.

## *17.1 Теоретические сведения*

Функция-элемент может быть объявлена как virtual . Ключевое слово virtual предписывает компилятору генерировать некоторую дополнительную информацию о функции. Если функция переопределяется в производном классе и вызывается с указателем (или ссылкой) базового класса, ссылающимся на представитель производного класса, эта информация позволяет определить, какой из вариантов функции должен быть выбран: такой вызов будет адресован функции производного класса.

Для виртуальных функций существуют следующие правила:

– виртуальную функцию нельзя объявлять как static;

– спецификатор virtual необязателен при переопределении функции в производном классе;

– виртуальная функция должна быть определена в базовом классе и может быть переопределена в производном.

Главное отличие виртуальной функции от просто перегруженной в том, какая функция будет вызываться при рассмотрении производного класса как базового.

***Пример.***

```
class Base
{
public:
	Base(){};
	void Print(){ cout<<"I’m a Base print"<<endl;}
	virtual void View(){ cout<<"I’m a Base view"<<endl;}
};

class Derived: public Base
{
public:
	Derived(){};
	void Print(){ cout<<"I’m a Derived print"<<endl;}
	void View(){ cout<<"I’m a Derived view"<<endl;}
};

int main ( )
{
	Base *A=new Base;
	Derived *B=new Derived;
	Base *C;
```

```
        A->Print();
        A->View();
        B->Print();
        B->View();
        C= (Base *)B;
        C->Print();
        C->View();
}
```

Результат:

«I’m a Base print»
«I’m a Base view»
«I’m a Derived print»
«I’m a Derived view»
«I’m a Base print»
«I’m a Derived view»

Таким образом, видим, что виртуальные функции позволяют всегда работать с теми функциями, которые специфичны именно для используемого класса, даже когда рассматриваем его как базовый.

### *17.2 Индивидуальные задания*

Разработать программу с использованием полиморфизма классов согласно вариантам задания (рисунок 17.1).

| *Вариант* | *Задание* |
|---|---|
| 1 | Разработать программу с использованием наследования классов, реализующую классы:<br>  <br><br>Используя чистые виртуальные функции, вывести на экран характеристики каждого из объектов |
| 2 | Разработать программу с использованием наследования классов, реализующую классы:<br>  <br><br>Используя чистые виртуальные функции, вывести на экран характеристики каждого из объектов |

Рисунок 17.1 – Исходные данные

***Контрольные вопросы***

1 Что такое полиморфизм? Какие виды полиморфизма бывают?

2 Объясните механизм реализации полиморфизма с помощью виртуальных функций.

3 Можно ли виртуальную функцию объявить как static?

# Список литературы

1 **Ашарина, И. В.** Язык C++ и объектно-ориентированное программирование в C++. Лабораторный практикум: учебное пособие / И. В. Ашарина, Ж. Ф. Крупская. – Москва: Горячая линия – Телеком, 2020. – 232 с.

2 **Павловская, Т. А.** C/C++. Программирование на языке высокого уровня: учебник для вузов / Т. А. Павловская. – Санкт-Петербург: Питер, 2020. – 432 с.