**ЧАСТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

 **ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«СТАВРОПОЛЬСКИЙ МНОГОПРОФИЛЬНЫЙ КОЛЛЕДЖ»**

**«Объектно-ориентированное программирование»**

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам по дисциплине

для обучающихся по специальности

 «Программирование в компьютерных системах»

**5 семестр**

Ставрополь 2022

Методические указания составлены в соответствии с ФГОС СПО по специальностям «Программирование в компьютерных системах» и с учетом современных требований к подготовке техников в области информационных технологий. Они содержат материал лабораторных работ первого семестра изучаемого курса. При этом основное внимание уделяется исследованию правил записи выражений, базовых типов данных, операций, циклов, массивов, указателей и связанных списков языка программирования С.

Рекомендуются студентам специальностей «Программирование в компьютерных системах» при изучении курса «Объектно-ориентированное программирование».

Составитель: Харченко И.В.

|  |  |
| --- | --- |
| Тема | Количество часов |
| Лабораторная работа 1. Функции языка программирования С | 4 |
| Лабораторная работа 2. Простой класс и реализация принципа наследования в С++ | 4 |
| Лабораторная работа 3. Перегрузка операций и функций | 4 |
| Лабораторная работа 4. Виртуальные функции | 4 |
| Лабораторная работа 5. Шаблонные функции и классы | 6 |
| Лабораторная работа 6. Простейшие классы и объекты | 4 |
| Лабораторная работа 7. Разработка классов С++ | 4 |
| Лабораторная работа 8. Классы для работы с динамическими структурами данных | 4 |
| Лабораторная работа 9. Шаблоны классов С++ | 6 |
| Лабораторная работа 10. Наследование С++ | 6 |
| Лабораторная работа 11. Потоки, обработка исключительных ситуаций в C++ | 6 |
| Лабораторная работа 12. Работа с типом данных структура. | 4 |
| Лабораторная работа 13. Коллекции. Параметризованные классы. | 4 |
| Лабораторная работа 14. Использование регулярных выражений. | 4 |

**СОДЕРЖАНИЕ**

[Лабораторная работа 1. Функции языка программирования С 8](#_Toc63442481)

[Лабораторная работа 2. Простой класс и реализация принципа наследования в С++ 23](#_Toc63442482)

[Лабораторная работа 3. Перегрузка операций и функций 35](#_Toc63442483)

[Лабораторная работа 4. Виртуальные функции 43](#_Toc63442484)

[Лабораторная работа 5. Шаблонные функции и классы 49](#_Toc63442485)

[Лабораторная работа 6. Простейшие классы и объекты 53](#_Toc63442486)

[Лабораторная работа 7. Разработка классов С++ 57](#_Toc63442487)

[Лабораторная работа 8. Классы для работы с динамическими структурами данных 62](#_Toc63442488)

[Лабораторная работа 9. Шаблоны классов С++ 67](#_Toc63442489)

[Лабораторная работа 10. Наследование С++ 72](#_Toc63442490)

[Лабораторная работа 11. Потоки, обработка исключительных ситуаций в C++ 80](#_Toc63442491)

[Лабораторная работа 12. Работа с типом данных структура. 84](#_Toc63442492)

[Лабораторная работа 13. Коллекции. Параметризованные классы. 86](#_Toc63442493)

[Лабораторная работа 14. Использование регулярных выражений. 87](#_Toc63442494)

[Список рекомендуемой литературы 94](#_Toc63442495)

**ВВЕДЕНИЕ**

Объектно-ориентированное программирование — это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Целью освоения дисциплины «Объектно-ориентированное программирование» является приобретение студентами знаний, умений и навыков применения современных технологии объектно-ориентированного программирования для создания экономических информационных систем в их профессиональной деятельности; формирование у студентов теоретических знаний и практических навыков разработки программ в оконных операционных системах с использованием объектно-ориентированной технологии программирования.

Задачи дисциплины:

- сформировать у студентов четкое представление об объектно-ориентированном подходе к разработке программного обеспечения и средствах его реализации;

- отработать твердые практические навыки по использованию объектно-ориентированных языков и соответствующих систем программирования для разработки программного обеспечения информационных систем.

**Аппаратура и материалы**

Практикум выполняется в лаборатории программирования, оборудованном IBM-совместимыми персональными компьютерами под управлением операционной системы Windows. Необходимым является наличие установленной на персональные компьютеры интегрированной среды разработки программных приложений Microsoft Visual Studio.

**Указания по технике безопасности**

Указания по технике безопасности в штатном компьютерном классе разработаны и утверждены администрацией и доводятся каждому студенту под роспись. Дополнительных требований нет.

**Содержание отчета и его форма**

Практические занятия оформляются в отдельной тетради. Отчет по каждому занятию должен содержать тему, цель, задание и выполненные программы, описание которых должно включать:

1. Постановку задачи.
2. Математическую модель.
3. Блок-схему алгоритма решения задачи.
4. Листинг программы.
5. Пробный вариант решения задачи.

**Лабораторная работа 1. Функции языка программирования С**

**Цель занятия:** рассмотреть особенности объявления и определений функций, способы задания формальных параметров и типов возвращаемых данных, вызов функций, передачу аргументов по значению и ссылке, функции с переменным числом аргументов.

**ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

В основе всех программ на языке программирования **С** лежат одни и те же фундаментальные элементы – функции. Например, к системным функциям относятся **printf()**, **scanf\_s()**, **gets\_s()**, **putchar()** и др. *Функции* – это строительные элементы языка **С** и то место, в котором выполняется вся работа программы. Большие программы обычно состоят из нескольких пользовательских и ряда системных функций. *Функция* – самостоятельная единица программы. Функции повышают уровень модульности программы, облегчают ее чтение, внесение изменений и коррекцию ошибок. В частности, функция **main()** является обязательной для любой программы. Во всех программах **С** определяется единая внешняя функция с именем **main()**, служащая точкой входа в программу, т. е. первой функцией, выполняемой после запуска программы.

Ни одна программа в языке **С** не может обойтись без функций.

Функция в языке **С** играет ту же роль, что и подпрограммы или процедуры в других языках. Каждая функция имеет имя и список аргументов. По соглашению, принятому в языке **С**, при записи имени функции после него ставятся круглые скобки. Так можно легко отличить имена переменных от имен функций.

Рассмотрим модель программы, в которой, кроме функции **main()**, содержатся еще три функции. Код примера будет следующим:

|  |
| --- |
| **#include <stdio.h>****int main(void) /\* главная функция \*/****{ /\* Начало тела функции \*/****function1(); /\* вызов первой функции \*/****function2(); /\* вызов второй функции \*/****function3(); /\* вызов третьей функции \*/****} /\* конец тела функции main() \*/****/\* Начало определения первой функции \*/****function1() { /\* начало тела первой функции \*/****/\* Операторы первой функции \*/****/\* Конец тела первой функции \*/}****/\* Начало определения второй функции \*/****function2()****{ /\* Начало тела второй функции\*/****/\* Операторы второй функции \*/****/\* Конец тела второй функции\*/}****/\* Начало определения третьей функции \*/****function3()****{ /\* Начало тела третьей функции\*/****/\* Операторы третьей функции \*/****/\* Конец тела третьей функции\*/}** |

В представленной программе имеются четыре функции (**main()**, **function1()**, **function2()**, **function3()**), не содержащие аргументов. Позднее рассмотрим функции, которые имеют аргументы. Аргументы функции – это величины, которые передаются функции во время ее вызова. Аргумент, стоящий в операторе вызова функции, называется *фактическим параметром*, аргументы, стоящие в заголовке функции, – *формальными параметрами*. В языке **С** функция может возвращать значение в вызывающую программу посредством оператора **return**. Оператор возврата из функции в точку вызова имеет две формы:

**return;**

**return выражение;**

В общем виде функция выглядит следующим образом:

**возвр-тип имя-функции(список параметров)**

**{**

**Тело\_функции**

**}**

 **Тело\_функции** – это часть определения функции, ограниченная фигурными скобками и непосредственно размещенная вслед за заголовком функции. Оно может быть либо составным оператором, либо блоком. В языке **С** определения функций не могут быть вложенными, т. е. внутри одной функции нельзя объявить и расписать тело другой функции.

Возвращаемый тип функции **возвр-тип** определяет тип данного, возвращаемого функцией. Это могут быть, например, **int**, **float**, **double** и т. д. В случае когда функция ничего не возвращает, ей присваивается тип **void**.

Функция может возвращать любой тип данных, за исключением массивов. Список параметров – это список, элементы которого отделяются друг от друга запятыми. При вызове функции параметры принимают значения аргументов. Если у функции нет параметров, то такой пустой список можно указать в явном виде, поместив для этого в скобки ключевое слово **void**. Все параметры функции (входящие в список параметров) должны объявляться отдельно, причем для каждого из них надо указывать и тип, и имя. В общем виде список объявлений параметров должен выглядеть следующим образом:

**fun(тип имя\_перем1, тип имя\_перем2, …, тип имя\_перем N)**

Например:

**fun(int i, int j, float k, char str1, char str2)**

Рассмотрим пример программы с выводом сообщения не в главной функции **main()**, а в другой.

|  |
| --- |
| **#include <stdio.h>****#include <conio.h>****void printMessage (void)****{printf("\n\t hello, world\n");****return;****printf("\n\t 123\n");}****int main(void)****{printMessage();****printf("\n Press any key: ");****\_getch();****return 0;}** |

Результат выполнения программы показан на рис. 10.1.



Рис. 10.1. Вывод сообщения с помощью двух функций

Программа состоит из двух функций: **printMessage()** и **main()**. Выполнение программы всегда начинается с функции **main()**, которую называют также главной. Внутри нее происходит вызов функции **printMessage()** без параметров. При этом выполнение программы передается непосредственно вызванной функции. Внутри функции **printMessage()** выполняется только утверждение

 **printf("\n\t hello, world\n");**

В функции **printMessage()** есть также утверждение **printf("\n\t 123\n")**, но оно не выполняется, поскольку используется утверждение возврата из функции (**return**).

В языке **С** функция введена как один из производных типов.

Формальные параметры в определениях функций могут объявляться в форме прототипов, которые дают компилятору возможность тщательнее проверять типы аргументов. Если используются прототипы, то компилятор может обнаружить любые сомнительные преобразования типов аргументов, необходимые при вызове функции, если тип ее параметров отличается от типов аргументов. Компилятор также обнаружит различия в количестве аргументов, использованных при вызове функции, и в количестве параметров функции.

В общем случае прототип функции должен выглядеть таким образом:

*тип имя\_функции*(*тип имя\_пар*1, *тип имя\_пар*2, …, *тип имя\_парN*);

В приведенной программе прототип функции **printMessage()** не применялся, так как сама функция была объявлена до главной функции **main()**. Для переносимости **С**-кода в **С++** использование прототипа функции обязательно. Поэтому хорошим стилем программирования можно считать работу с прототипами функций, поскольку большие программы обычно состоят из нескольких функций, часто расположенных в различных файлах.

Рассмотренная ранее программа с использованием прототипа функции **printMessage()** будет выглядеть следующим образом.

|  |
| --- |
| **#include <stdio.h>****#include <conio.h>****//void printMessage (void);// прототип функции****int main(void)** **{ void printMessage (void); // прототип функции**  **printMessage(); // вызов функции**  **printf("\n Press any key: ");** **\_getch();** **return 0; }****// Определение функции****void printMessage (void)****{ printf("\n\t hello, world\n");** **return;** **printf("\n\t 123\n");}** |

В листинге программы показаны две возможности применения прототипа функции **printMessage()**. При этом сама эта функция объявлена после функции **main()**.

*Формальные параметры* функции определены в ее прототипе. При обращении к функции используются *фактические параметры*, называемые *аргументами функции*.

Список фактических параметров – это список выражений, количество которых равно количеству формальных параметров функции (исключение составляют функции с переменным числом параметров). Соответствие между формальными и фактическими параметрами устанавливается по их взаимному расположению в списках. Между ними должно наблюдаться соответствие по типам.

Синтаксис языка **С** предусматривает только один способ передачи параметров – передачу по значениям. Формальные параметры функции локализованы в ней, т. е. недоступны вне определения функции, и никакие операции над формальными параметрами в теле функции не изменяют значений фактических параметров.

Передача параметров по значению предусматривает следующие шаги.

1. При компиляции функции выделяются участки памяти для формальных параметров, т. е. формальные параметры оказываются внутренними объектами функции. При этом для параметров типа **float** формируются объекты типа **double**, а для параметров типов **char** и **short int** – объекты типа **int**. Если параметром является массив, то создается указатель на начало этого массива, и он служит представлением массива-параметра в теле функции.
2. Вычисляются значения выражений, использованных в качестве фактических параметров при вызове функции.
3. Значения выражений – фактических параметров заносятся в участки памяти, выделенные для формальных параметров функции.
4. В теле функции используются значения внутренних объектов-параметров, и результат передается в точку вызова функции как возвращаемое ею значение.
5. Никакого влияния на фактические параметры (на их значения) функция не оказывает.
6. После выхода из функции освобождается память, выделенная для ее формальных параметров.

Важно то, что объект вызывающей программы, использованный в качестве фактического параметра, не может быть изменен из тела функции. Но существует *косвенная возможность* изменять значения объектов вызывающей программы действиями в вызванной функции. Это реализуется с помощью указателя (указателей), когда в вызываемую функцию передается адрес любого объекта из вызывающей программы. С помощью выполняемого в тексте функции разыменования указателя осуществляется доступ к адресуемому указателем объекту из вызывающей программы. Тем самым, не изменяя параметра (указатель-параметр постоянно содержит только адрес одного и того же объекта), можно изменять объект вызывающей программы.

Массивы и строки также могут быть параметрами функции. В этом случае внутрь функции передается только адрес начала массива. Тогда можно в качесстве параметра использовать указатель. Приведем два равноправных прототипа функций:

**float fun(int n, float A[ ], float B[ ]);**

**float fun(int n, float \*a, float \*b);**

Поскольку массив передается в функцию как указатель, внутри функции можно изменять значения элементов массива – фактического параметра, определенного в вызывающей программе. Это возможно и при использовании индексирования, и при разыменовании указателей на элементы массива.

В языке **С** существует возможность создавать функции, число аргументов которых не определено – функции с переменным числом аргументов. При этом следует указать только количество аргументов. Пример прототипа такой функции:

**int fun(int n, …);**

Многоточие  означает, что функция получает переменное число аргументов любого типа. Многоточие должно всегда находиться в конце списка параметров.

Макросы и определения заголовочного файла переменных аргументов **stdarg.h** (табл. 10.1) предоставляют программисту средства, необходимые для построения функций со списком аргументов переменной длины.

Примеры обращений к функции с фактическими аргументами:

|  |
| --- |
| **double k;****double v1 = 1.5,** **v2 = 2.5,** **v3 = 3.5;****// Первый вариант, где 3 – количество аргументов****k = fun(3,v1, v2, v3);****// Второй вариант, где 0.0 – завершающий нуль списка аргументов****k = fun(v1, v2, v3, 0.0);** |

|  |  |
| --- | --- |
|  | *Таблица 10.1* |
| *Макросы заголовочного файла* ***stdarg.h*** |
| Идентификатор | Объяснение |
| **va\_list** | Тип, предназначенный для хранения информации, необходимой макросам **v\_start**, **va\_arg** и **va\_end**. Чтобы получить доступ к аргументам в списке переменной длины, необходимо объявить объект типа **va\_list** |
| **va\_start** | Макрос, который вызывается перед обращением к аргументам списка переменной длины. Он инициализирует объект, объявленный с помощью **va\_list**, для использования макросами **va\_arg** и **va\_end** |
| **va\_arg** | Макрос, расширяющийся до выражения со значением и типом следующего аргумента в списке переменной длины. Каждый его вызов изменяет объект, объявленный с помощью **va\_list**, так, что объект указывает на следующий аргумент списка |
| **va\_end** | Макрос обеспечивает нормальный возврат из функции, на список аргументов которой ссылается макрос **va\_start** |

**ПРАКТИЧЕСКАЯ ЧАСТЬ**

**Задание к лабораторной работе**

*Пример* 1. Написать программу сортировки по возрастанию заданного массива случайных чисел, равномерно распределенных в интервале [–6; 6], с помощью вспомогательной функции.

Программный код решения примера

|  |
| --- |
| **#include <stdio.h>****#include <conio.h>****#include <stdlib.h>****#include <time.h>****#define MAX 10****// Прототип функции с формальными параметрами****void sort(double arr[], int n);****int main (void)** **{ double M[MAX];** **int i, size = MAX;** **long int L;** **time\_t t;** **srand((unsigned)time(&t));** **for (i = 0; i < MAX; ++i)** **M[i] = 12.0\*rand()/RAND\_MAX - 6.0;** **printf("\n\t The original array:\n");** **for (i = 0; i < MAX; ++i) {printf("\t%8.4f\n", M[i]);}** **sort(M, size);// обращение к функции с фактическими параметрами** **// Распечатка отсортированного массива** **printf("\n\t After sorting: \n");** **for (i = 0; i < MAX; ++i)** **printf("\t%8.4f\n", M[i]);** **printf("\n Press any key: ");** **\_getch();** **return 0;}** **// Пользовательская функция сортировки**  **void sort(double Array[], int m)** **{ int i, j;** **double tmp;** **for (i = 0; i < m-1; ++i)** **for (j = 0; j < m-i-1; ++j)** **if (Array[j+1] < Array[j])** **{ tmp = Array[j];** **Array[j] = Array[j+1];** **Array[j+1] = tmp;}** **}** |

Следует обратить внимание на формальные параметры в самой функции **sort()** и в ее прототипе: они имеют разные имена, но одинаковые типы. Фактические параметры, или аргументы, функции **sort()** в вызывающей программе (в теле функции **main()**) имеют имена, не связанные с именами формальных параметров.

Заполнение массива случайными числами производится с помощью библиотечной функции **rand()** и макроопределения **RAND\_MAX**.

Возможный результат выполнения программы представлен на рис. 10.2.



Рис. 10.2. Пример сортировки числового массива

*Задание* 1

1. Используйте программу без прототипа функции **sort()**.
2. С помощью вспомогательной функции произведите сортировку массива по убыванию.
3. С помощью вспомогательной функции произведите сортировку целых чисел из заданного интервала [X; 10X], где Х – номер компьютера, на котором выполняется лабораторная работа.
4. Создайте функцию умножения исходного массива как матрицы-столбца на отсортированный массив как матрицу-строку. Выведите на консоль результат перемножения.
5. Создайте функцию умножения исходного массива как матрицы-строки на отсортированный массив как матрицу-столбец. Выведите на консоль результат перемножения.

*Пример* 2. Написать программу вычисления квадратного корня числа по методу Ньютона – Рафсона с использованием функции расчета квадратного корня числа и функции расчета абсолютного значения числа.

При вычислении квадратного корня из числа следует помнить, что подкоренное выражение не должно быть отрицательным.

*Алгоритм метода Ньютона – Рафсона* для вычисления квадратного корня числа.

*Шаг* 1. Выбрать приблизительное значение 1.

*Шаг* 2. Если |*guess*2 – *x*| < *e*, перейти к шагу 4 (англ. *guess* – «предположение»).

*Шаг* 3. Установить приблизительное значение, равное (*x*/*guess* + *guess*)/2, и перейти к шагу 2.

*Шаг* 4. Считать приблизительное значение квадратным корнем числа.

Программный код решения примера

|  |
| --- |
| **#include <stdio.h>****#include <conio.h>** **// Функция абсолютного значения числа** **double absValue(double x)** **{ if (x < 0) x = -x; return (x); }** **// Функция расчета квадратного корня из числа****double squareRoot (double x)** **{ const double epsilon = 0.000001;** **double guess = 1.0; //Начальное приближение** **while (absValue(guess\*guess - x) >= epsilon)** **guess = (x/guess + guess)/2.0;** **return (guess);}****int main (void)** **{ double result, X;** **printf("\n\t Enter a number: ");** **scanf\_s("%lf", &X);****// Обращение к функции с фактическим параметром**  **result = squareRoot(X);****printf("\n\t Square root of \"%1.4f\" is: %1.8f\n", X, result);** **printf("\n Press any key: ");** **\_getch();** **return 0;}** |

В программе используются три функции: **main()**, **absValue()**, **squareRoot()**. Вспомогательные функции расположены в определенном порядке: сначала **absValue()**, а потом **squareRoot()**. В первую передается значение **guess**, которое вычисляется в функции **squareRoot()**, находящейся в условии оператора цикла **while**. Когда условие для оператора цикла будет ложным, т. е. когда значение корня будет меньше заданного числа **epsilon**, полученное значение возвращается в вызывающую функцию **main()**.

В программе использована переменная **epsilon**, определенная с помощью спецификатора **const**, что делает ее неизменной. При таком объявлении компилятор определяет ее как константное значение. Таким переменным нельзя присваивать значения в программе, их нельзя инкрементировать или декрементировать.

Возможный результат выполнения программы показан на рис. 10.3.



Рис. 10.3. Результат итерационного извлечения квадратного корня из двух

*Задание* 2

1. В функции **absValue()** примените оператор условия **?** вместо оператора **if** без определения дополнительной переменной.
2. В главной функции **main()**предусмотрите проверку ввода неотрицательного числа.
3. В программу введите прототипы функций.
4. Зафиксируйте результат вычисления квадратного корня от числа Х, где Х – номер компьютера, на котором выполняется лабораторная работа.
5. В качестве начального приближения возьмите 1,5Х, где Х – номер компьютера, на котором выполняется лабораторная работа.
6. В программе примените тип **float** вместо типа **double**.

*Пример* 3. Написать программу поиска максимального элемента среди минимальных элементов строк двухмерного целочисленного массива.

Условие примера соответствует поиску *максмина* в двухмерном массиве, т. е.



где  – элементы матрицы *А*.

Алгоритм поиска максминазаключается в следующем. Сначала предполагаем, что максимальным элементом является начальный элемент первой строки **A[0][0]**, затем заменяем его минимальным элементом этой же строки, т. е. теперь минимальный элемент первой строки принимается за искомый максимум. Последовательно просматривая остальные строки, находим в каждой из них минимальный элемент. Если окажется, что минимальный элемент текущей строки больше текущего максимума, то он принимается за максимальный. Результатом поиска являются значение максимального элемента и его индексы (номер строки и номер столбца).

Программный код решения примера

|  |
| --- |
| **// Поиск максмина****#include <stdio.h>****#include <conio.h>****#include <stdlib.h>****#include <time.h>****#define n 6****#define m 7****const int N = 100;****int main (void)** **{** **//Прототип функции****int MaxMin(int A[][m], int nn, int mm, int \*imax, int \*jmax);** **int i, j, A[n][m], max, imax, jmax;** **time\_t t;** **srand((unsigned)time(&t));** **//Заполнение матрицы целыми случайными числами из [-N; N-1]** **for (i = 0; i < n; ++i)** **for (j = 0; j < m; ++j)** **A[i][j] = (rand() % N) - N;** **// Распечатка матрицы** **printf("\n\t The original matrix A(%d\*%d):\n\n", n, m);** **for (i = 0; i < n; ++i){** **printf("\t");** **for (j = 0; j < m; ++j)** **printf("%4d", A[i][j]);** **printf("\n");** **}** **max = MaxMin(A, n, m, &imax, &jmax);****printf("\n\t Result: MaxMin = A[%d][%d] = d\n",imax+1,jmax+1,****max);** **printf("\n Press any key: ");** **\_getch();** **return 0;****}****// Функция поиска максмина****int MaxMin(int A[][m], int nn, int mm, int \*imax, int \*jmax) {** **int i, j, min, max, imin, jmin;** **max = A[0][0];** **\*imax = 0; \*jmax = 0;** **for (j = 1; j < mm; ++j)** **if (A[0][j] < max)** **{max = A[0][j]; \*imax = 0; \*jmax = j;}** **for (i = 1; i < nn; ++i) {** **min = A[i][0]; imin = i; jmin = 0;** **for (j = 1; j < mm; ++j)** **if (A[i][j] < min)** **{min = A[i][j]; imin = i; jmin = j;}** **if (max < min)** **{max = min; \*imax = imin; \*jmax = jmin;}** **}** **return max;****}** |

В объявлении функции **MaxMin()** для матрицы явно указывается только второй размер – количество столбцов. Указатели **\*imax**, **\*jmax** используются для передачи в вызывающую функцию индексов найденного максмина. Возвращение самого максмина осуществляется по значению с помощью оператора **return**, т. е. **return max**.

Возможный результат выполнения программы представлен на рис. 10.4.



Рис. 10.4. Пример вычисления максмина двухмерного массива

*Задание* 3

1. Подсчитайте число итераций, используемых при поиске максмина.
2. В объявлении матрицы **MaxMin()** (и ее прототипа) примените указатель на одномерный массив, например **int \*a** вместо **int A[ ][m]**.
3. Число строк матрицы задайте как 3Х, где Х – номер компьютера, на котором выполняется лабораторная работа.
4. Размерность анализируемого массива (матрицы) задайте с клавиатуры. С клавиатуры задайте также интервал генерирования случайных чисел функцией **rand()** в виде [–2Х; 2Х], где Х – номер компьютера, на котором выполняется лабораторная работа.
5. Напишите функцию определения субмаксмина, т. е. максимального числа среди минимальных чисел строк двухмерного массива после найденного максмина.
6. Определите абсолютные значения максимума и минимума сформированной матрицы, для которой находится максмин.

*Пример* 4. Написать программу поиска одного элемента в линейной неупорядоченной таблице по совпадению ключа на основе заграждающего элемента. Таблицу описать в виде одномерного массива целых чисел.

В линейной таблице элементы располагаются друг за другом, т. е. для каждого из них существуют отношения порядка. Линейные таблицы в оперативной памяти компьютера представлены массивами или линейными связанными списками.

Поиск одного элемента в неупорядоченной таблице по заданному условию осуществляется последовательным просмотром элементов или до нахождения искомого, т. е. до выполнения условия поиска, или до конца таблицы, если искомый элемент не обнаружен. Возвращаемыми значениями являются адрес (индекс) или значение элемента либо признак его отсутствия.

В случае применения *заграждающего элемента* последняя запись таблицы запоминается, а после завершения поиска восстанавливается в таблице. В последний элемент массива (когда таблица представляется в виде одномерного массива данных) заносится ключ поиска, и образуется так называемый *заграждающий элемент*. Теперь на каждом шаге поиска осуществляется только одно сравнение, а сам поиск продолжается до нахождения элемента с заданным ключом. Если искомого элемента в исходной таблице не было, то поиск закончится на заграждающем элементе. Таким образом, его использование при наличии числовых ключей существенно сокращает количество сравнений.

Программный код решения примера

|  |
| --- |
| **#include <stdio.h>****#include <conio.h>****#define size 10****// Таблица числовых элементов****int A[size] = {1,-7,3,4,8,-5,-2,6,0,9};****int main (void) {**  **int i, ind, key;** **int search(int A[], int n, int key);// прототип функции** **printf("\n\t The original array:\n\t");** **for (i = 0; i < size; ++i) printf("%3d", A[i]);** **// Поиск ключа соответствия** **printf("\n\n\t Search for key matches.\n");** **// Ввод ключа поиска** **printf("\t Key in your search: ");** **scanf\_s("%i", &key);** **ind = search(A, size, key); // обращение к функции** **if (ind == -1)** **printf("\n\t Element \"%d\" could not be found\n", key);** **else** **printf("\n\t Element \"%d\" index is %d\n", key, ind+1);** **printf("\n Press any key: ");** **\_getch(); return 0;****}** **int search(int A[], int n, int key) {** **int i = 0, r;** **r = A[n-1];** **A[n-1] = key;// заграждающий элемент** **while (A[i] != key)** **i++;** **A[n-1] = r; // восстановление последнего элемента** **if ((i == n - 1) && (r != key))** **return -1; // отсутствие элемента** **else** **return i; // успешный поиск****}** |

В программе использовано внешнее объявление одномерного массива с инициализацией. Слово «внешний» означает расположение объявления вне функций, напрямую с ключевым словом **extern**  оно не связано.

*Задание* 4

1. Подсчитайте число итераций при поиске элемента по заданному ключу.
2. Напишите программу без заграждающего элемента. Подсчитайте число итераций при поиске элемента по заданному ключу. Сравните с числом итераций при поиске с заграждающим элементом.
3. Сформируйте динамический одномерный массив размерностью 12Х и заполните его целыми случайными числами, распределенными равномерно из интервала [0; 20X], где Х – номер компьютера, на котором выполняется лабораторная работа. Ввод ключа для поиска выполните с клавиатуры. Поиск выполните с заграждающим элементом.

 Возможный результат выполнения программы показан на рис. 10.5.



Рис. 10.5. Пример поиска элемента в массиве по заданному ключу

1. С учетом предыдущего пункта задания ввод ключа задайте случайным образом. Если по этому ключу элемент не найден, то программным путем организуйте повторный поиск с заданием нового случайного ключа. В случае если при 5Х-кратном поиске элемент не найден, выведите сообщение об отсутствии искомого элемента по всем заданным ключам. Поиск выполнить с заграждающим элементом.

*Пример* 5. Написать программу выделения слов из символьной строки, когда они разделены пробелами, и поместить каждое слово в отдельной строке свободного (вспомогательного) массива.

Программный код решения примера

|  |
| --- |
| **#define \_CRT\_SECURE\_NO\_WARNINGS****#include <stdio.h>****#include <conio.h>****#include <stdlib.h>****#include <string.h>****#define N 123****#define M 40****int wordstr(char \*c, char \*\*FA) {** **int Ln, n = 0, i = 0;** **char \*wr;** **char \*tempstr;** **Ln = strlen(c);** **tempstr = (char \*)calloc(Ln + 1, sizeof(char));** **strcpy(tempstr, c);** **wr = strtok(tempstr, " ");** **Ln = strlen(wr);** **FA[i] = (char \*)calloc(Ln + 1, sizeof(char));** **strcpy(FA[i++], wr);** **n++;** **while (wr = strtok(NULL, " "))**  **{** **Ln = strlen(wr);** **FA[i] = (char \*) calloc(Ln + 1, sizeof(char));** **strcpy(FA[i++], wr);** **n++;** **}** **free(tempstr);** **return n;****}****int main (void)** **{** **int i, n;** **char \*Farr[M], str[N];** **printf("\n\t Enter the string of characters:\n >> ");** **gets\_s(str, N-1);** **printf("\n\t The original string of characters:\n");** **printf(" %s\n", str);** **//Обращение к функции обработки строки** **n = wordstr(str, Farr);** **//Распечатка слов, помещенных в свободный массив** **printf("\n\t The characters in a free array:");** **for (i = 0; i < n; ++i)** **printf("\n\t %s", Farr[i]);** **//Освобождение памяти, занятой выделенными словами** **for (i = 0; i < n; ++i)** **{free(Farr[i]); Farr[i] = NULL;}** **printf("\n\n Press any key: ");** **\_getch();** **return 0; }** |

В программе использована библиотечная функция **strtok()**, которая выделяет слова из строки. При этом для каждого выделенного слова получаем динамическую память с помощью функции **calloc()**. Адрес выделенной памяти помещаем в соответствующий указатель массива свободных строк **FA[]**. Обратите внимание, что в функции **calloc()** вводится число на единицу больше, чем длина строки. Это сделано для учета символа завершения строки **'\0'** (или одного слова).

Для исключения предупреждений о безопасной работе с функциями **strcpy()**, **strtok()** в MS Visual Studio в программе используется директива **#define \_CRT\_SECURE\_NO\_WARNINGS**.

Результат выполнения программы представлен на рис. 10.6.



Рис. 10.6. Пример выделения слов из строки

*Задание* 5

1. В программу включите обработку символов: запятая '**,**', точка с запятой '**;**', точка '**.**' в качестве разделителей слов.
2. Используйте символ пробела для разделения слов «фамилия, имя, год поступления в вуз».
3. В программу включите прототип функции **wordstr()**.
4. Изучите возможность применения в программе функций **strcpy\_s()** и **strtok\_s()**.

*Пример* 6. Написать программу расчета суммы и среднего арифметического числовых произвольного количества данных. В качестве ключевых слов для выбора варианта расчета принять *mean* (среднее) и *sum* (сумма).

В данном примере необходимо использовать функцию с переменным числом агрументов. Среднее арифметическое (*m*) рассчитывается по формуле



где *n* – количество числовых данных, *xi* – текущее число.

Программный код решения примера

|  |
| --- |
| **#include <stdio.h>****#include <conio.h>****#include <string.h>****#include <stdarg.h>****#include <float.h>****/\* Прототип функции с переменным числом аргументов\*/****double varfun(char str[], double v1, ...);** **int main(void)****{** **double v1 = 10.0,** **v2 = 2.5,** **control;** **char str[121];** **printf("\n Enter one of the key words 'mean' or 'sum': ");** **gets\_s(str, 120);** **control = varfun(str, v1, v2, 7.5, 0.0);** **if (control < DBL\_MAX)** **printf("\n Result: %lf\n", control);** **else** **printf("\n Invalid input keyword.\n");** **printf("\n\n ... Press any key: ");** **\_getch();** **return 0;}****/\* Определение функции с переменным числом аргументов \*/****double varfun( char str[], double v1, ...) {****/\* Указатель на переменные списка аргументов \*/** **va\_list parg;**  **double sum = v1;** **double value = 0.0; /\* значение аргумента \*/** **int count = 1; /\* начальное количество аргументов \*/** **int k, p;** **char \*ch = "mean";** **char \*ch2 = "sum";** **// Лексиграфическое сравнение строк** **k = strcmp(str, ch);** **if ( !k ) // k == 0** **p = 1;** **else if ( k ) // k != 0** **{** **k = strcmp(str, ch2);** **if ( !k )** **p = 2;** **else** **p = 0;** **}** **if ( p == 1 ) {** **va\_start(parg,v1); /\* инициализация указателя parg \*/** **/\* Просмотр списка аргументов\*/** **while( (value = va\_arg(parg, double)) != 0.0)** **{** **// Суммирование числовых аргументов функции varfun()** **sum += value;** **count++;** **}** **/\* Завершение процесса счтывания аргументов \*/** **va\_end(parg);**  **return sum/count;** **}** **if ( p == 2 )** **{** **va\_start(parg,v1);** **while( (value = va\_arg(parg, double)) != 0.0)** **{** **sum += value;** **count++;** **}** **va\_end(parg);**  **return sum;** **}** **return DBL\_MAX;****}** |

Возможный вариант выполнения программы показан на рис. 10.7.



Рис. 10.7. Пример выполнения программы

*Задание*6

1. Напишите функцию с явным указанием количества аргументов.
2. Включите в программу расчет исправленной выборочной дисперсии (*D*), которая рассчитывается по формуле



где *m* – среднее арифметическое заданных чисел.

**Лабораторная работа 2. Простой класс и реализация принципа наследования в С++**

**Цель и содержание**: Приобрести навыки создания простых классов в языке С++, а также их иерархий.

**Теоретическое обоснование**

*Класс* представляет собой уникальное множество элементов и операций, а также преобразований, используемых для создания, манипуляции и уничтожения объектов, наделенных свойствами класса.

Класс − это пользовательский *тип данных*. Тип данных − есть конкретное представление какого-либо понятия некоей предметной области. Например, встроенный тип *float* является представлением математического понятия вещественного числа. Новый тип (класс) создается, чтобы отразить в программе сущность предметной области, которую невозможно представить с помощью одного из встроенных типов. В качестве примеров таких сущностей можно привести: вектор в математике, кортеж, соединение − в теории баз данных, работник, документ − в автоматизированных системах управления и т. д.

Внутри класса могут содержаться функции и переменные, называемые *членами* класса. Переменные внутри класса называются *полями,* а функции − *методами*. Для членов класса можно определять различный уровень доступа к ним. Для этого используются *модификаторы доступа:*

* **private** − закрытый, доступ к члену возможен только из методов данного класса;
* **public** − открытый, доступ к члену возможен из любой части программы;
* **protected** − защищенный, доступ к члену возможен только из методов данного класса и его наследников.

Идея ограничения доступа к членам класса состоит в том, чтобы скрыть от пользователя класса (программиста) внутренние детали его описания и реализации (с помощью модификатора private), взамен предоставив «интерфейс» для использования класса, то есть набор только тех полей и методов, которые необходимы для использования данного класса. Такие поля и методы помечаются модификатором public. Модификатор protected применяется при построении иерархий классов.

*Наследование* – это механизм, посредством которого можно описать класс на основе уже существующего класса (или нескольких классов), при этом поля и методы существуещего класса наследуются новым.

Когда один класс наследуется другим, класс, свойства которого должны быть унаследованы, называется *базовым* (base class), а класс, получающий эти свойства – *производным* (derived class) или *классом-наследником*. Обычно процесс наследования начинается с определения базового класса, как имеющего те качества, которые являются наиболее общими для всех производных от него классов. Производный класс наследует все черты базового класса и добавляет те свойства, которые характерны только для него.

Два или более класса, связанных отношением наследования называются *иерархией классов*. Приведем простой пример иерархии, состоящей из двух классов (определения методов не показаны).

/\* Базовый класс \*/

class Base

{

private:

 int a;

protected:

 int b;

public:

 void SetA(int num);

 int GetA();

 void SetB(int num);

 int GetB();

};

/\* Производный класс \*/

class Derived : public Base

{

private

 int c;

public

 void SetС(int num);

 int GetС();

};

int main()

{

 Base baseObject;

 Derived devivedObject;

}

Синтаксис наследования определен следующим образом: после имени класса ставится символ «:» (двоеточие), далее следует тип наследования (в данном случае public) и имя класса, от которого производится наследования. В С++ существует 3 вида наследования:

* **public −** наследование, при котором модификаторы доступа наследуемых полей и методов базового класса остаются в производном классе без изменений.
* **protected −** наследование**,**  при котором все модификаторы остаются без изменения, кроме модификатора public, который меняется на protected
* **private-наследование −** наследование**,**  при котором все модификаторы доступа меняются на private.

Обратите внимание, что в данном примере используется модификатор доступа *protected*, который устанавливает, что доступ к полю **b** класса **Base** возможен как из класса **Base**, так и из его наследника **−** класса **Derived,** в отличие от закрытого поля **а**класса **Base**, доступ к которому возможен только из класса **Base.**

Если базовый класс содержит конструктор с непустым списком параметров, то он должен быть вызван в производном классе. Список параметров для конструктора базового класса указывается в конструкторе производного класса.

Например, опишем иерархию классов « Человек: Студент».

class Person

{

protected:

 string firstName;

 string lastName;

public:

 Person(string& firstName, string& lastName);

};

Person::Person(string& firstName, string& lastName)

{

 this->firstName = firstName;

 this->lastName = lastName;

}

class Student : public Person

{

private:

 int studyYear;

 string department;

public:

 Student(string& firstName, string& lastName, string& department,

 int studyYear);

};

**Пример выполнения задания**

**Задание 1.** Создать простой класс, описывающий понятие «Автомобиль».

Закрытыми членами в данном случае будут двигатель и прочие «внутренности» автомобиля, а открытым интерфейсом для его использования — руль, педали, рычаг переключения скоростей и приборная панель.

 Синтаксис объявления класса следующий:

 **class** SampleClass //имя класса

 {

 /\* Методы и поля класса \*/

 **private**:

 int number; // поле класса

 **public**:

 int GetNumber(); // метод класса

 void SetNumber(int num);

 };

В классе SampleClass содержится одно закрытое поле *number* и два открытых метода GetNumber( ) и SetNumber(int). Заметим, что методы пока только объявлены, то есть в классе содержатся только их прототипы (сигнатуры), а сами действия производимые ими ещё не определены. Синтаксис определения методов следующий:

int SampleClass::GetNumber()

{

 return number;

}

void SampleClass::SetNumber(int num)

{

 if(number < 10)

 number = num;

 else

 cout << «Некорректное значение»;

}

Подобно определению функции, определение метода начинается с описания типа возвращаемого значения (в данном случае int и void). Далее следует имя класса, к которому относится метод (SampleClass), затем оператор «::» − оператор расширения диапазона и имя метода (GetNumber, SetNumber) со списком его формальных параметров.

Распространенной практикой в программировании на языках высокого уровня, поддерживающих ООП, является создание пары открытых методов для доступа к закрытому полю. Имена этих методов обычно начинаются с префиксов Set (установить) и Get (получить), за которыми следует имя переменной, к которой осуществляется доступ. Эти два метода соответственно устанавливают и возвращают значение закрытого поля. Такой подход применяется для гибкости, т. к. в момент инициализации поля можно осуществить, например, проверку на корректность значения или любое другое действие.

Само по себе объявление класса, определение его полей и методов не произведет в вашей программе никаких действий. Для этого необходимо создать *объект* класса. Объект есть переменная типа класса. Создание объекта класса происходит подобно созданию переменной встроенного типа:

int main()

{

 int a;

 double b;

 SampleClass object1;

 SampleClass object2;

}

В данном примере, помимо переменных а и b, в памяти будут созданы 2 объекта класса SampleClass, каждый из которых будет обладать структурой и поведением, описанным в классе SampleClass. То есть объекты object1 и object2 будут обладать каждый своим закрытым полем number и методами GetNumber и SetNumber.

Доступ к полям и методам объектов класса осуществляется с помощью оператора «.»:

int main()

{

 int n;

 SampleClass object;

 object.SetNumber(8); /\* Вызываем метод \*/

 n = object.GetNumber(); /\* n равно 8 \*/

 object.number = 5; /\* Ошибка! поле number является закрытым \*/

}

Начальная инициализация полей объекта класса с помощью методов типа SetNumber не является лучшей идеей, т.к. программист может либо забыть инициализировать поле, либо сделать это дважды, что приведет к неправильной работе программы. Идеальным решением было бы инициализировать поля во время создания объекта в памяти. Для этого в классе определятся специальный метод, называемый *конструктором.* Как и обычные методы, конструктор может иметь список параметров, но в отличие от них, конструктор не имеет возвращаемого значения и его имя всегда совпадает с именем класса:

 **class** SampleClass

 {

 **private**:

 int number;

 **public**:

 SampleClass(int num); /\* Объявление конструктора \*/

 int GetNumber();

 void SetNumber(int num);

 };

 /\* Определение конструктора \*/

 SampleClass::SampleClass(int num)

 {

 number = num;

 }

 int main()

 {

 /\* Использование конструктора \*/

 SampleClass object1(4);

// сокращенная форма

 SampleClass object2 = SampleObject(6); // полная форма

 int a = object1.GetNumber(); // a = 4;

 int b = object2.GetNumber(); // b = 6;

 }

**Задание 2.** Описать иерархию классов «Студент  Человек».

class Person

{

protected:

 string firstName;

 string lastName;

public:

 Person(string& firstName, string& lastName);

};

Person::Person(string& firstName, string& lastName)

{

 this->firstName = firstName;

 this->lastName = lastName;

}

class Student : public Person

{

private:

 int studyYear;

 string department;

public:

 Student(string& firstName, string& lastName, string& department,

 int studyYear);

};

Student::Student( string& firstName, string& lastName, string& department,

 int studyYear) : Person(firstName, lastnName)

{

 this->department = department;

 this->studyYear = studyYear;

}

**Методика и порядок выполнения лабораторной работы**

1. Изучить теоретический материал по лекциям, дополнительной литературе.
2. Изучить теоретическое обоснование лабораторной работы.
3. Ответить на контрольные вопросы.
4. Выполнить индивидуальное задание 1.
5. Выполнить индивидуальное задание 2.
6. Составить отчет о лабораторной работе.

**Задание к лабораторной работе**

**Задание 1.** Создать класс, описывающий следующее понятие, и продемонстрировать работу с ним.

Таблица 1. – Варианты индивидуального задания 1

|  |  |  |  |
| --- | --- | --- | --- |
| № варианта | Понятие | Атрибуты | Методы |
| *1* | *2* | *3* | *4* |
| 1 | Студент | ФИО, год рождения, группа, оценки за последнюю сессию | вычисление стипендии на основании последней сессии,выдача справки об успеваемости |
| 2 | Персональный компьютер | типы монитора, процессора, видео карты, оперативной памяти и их стоимость | подсчет стоимости персонального компьютера,вывод типов и стоимости всех комплектующих |
| 3 | Дата | день, месяц, год | добавить к дате указанное количество дней,преобразовать в строку, содержащую дату |
| 4 | Треугольник | длины сторон  | расчет площади, расчет периметра |
| 5 | Прямоугольник | длины сторон | расчет площади, расчет периметра |
| 6 | Круг | Радиус | расчет площади, расчет длины окружности |
| 7 | Человек | возраст, имя, фамилия, отчество, пол | предположить социальный статус с учетом пола (ребенок, школьник/школьница и т.п.),вывод информации о человеке на экран |
| 8 | Студент | фамилия, группа, оценки по предметам за последнюю сессию | вывод всех оценок по предметам, сдаваемым в сессию,расчет среднего балла |

Продолжение таблицы 1

|  |  |  |  |
| --- | --- | --- | --- |
| *1* | *2* | *3* | *4* |
| 9 | Рабочий | фамилия, стаж, часовая заработная плата, количество отработанных часов | расчет заработной платы за отработанное время; в зависимости от стажа начисляется премия (при стаже до 1 года 0%, до 3 лет 5%, до 5 лет 8%, свыше 10 лет 15%),вывод информации о рабочем на экран |
| 10 | Телефон | номер, телефонная книга на 10 номеров, название телефона и фирма производитель | добавление/удаление номера в телефонной книге, поиск в телефонной книге номера по имени |
| 11 | Телефон | собственный номер, 10 последних набранных номеров и длительность разговора для каждого, название телефона и фирма производитель | набор номера, вывод списка звонков,подсчет суммарного времени звонков |
| 12 | Банкомат | сумма денег, хранящихся в банкомате для 3 и более разных валют, адрес банкомата, название банка | «выдача» денежных средств в любой валюте клиенту,вывод справки о хранящейся валюте |
| 13 | Автомат по продаже прохладительных напитков | стоимость 3-х и более напитков, запас каждого напитка, сумма полученных денег | вывод ассортимента напитков, их стоимости,«продажа» покупателю напитка,вывод суммы денег, находящихся в автомате |

Продолжение таблицы 1

|  |  |  |  |
| --- | --- | --- | --- |
| *1* | *2* | *3* | *4* |
| 14 | Вектор в декартовых координатах | координаты х, у | ввод вектора, вывод вектора, скалярное произведение двух векторов |
| 15 | Квадратная целочисленная матрица | матрица 3 х 3 | ввод/вывод матрицы,вычисления следов матрицы и ее определителя |
| 16 | Отрезок, задаваемый координатами своих концов | координаты концов отрезка | ввод/ вывод координат отрезка,определение длины отрезка  |
| 17 | Вектор в декартовой системе координат | координаты вектора х, у | ввод вектора,сложение/вычитание векторов |
| 18 | Вектор в декартовой системе координат | координаты вектора х, у | ввод вектора,умножение вектора на скаляр |
| 19 | Прямоугольная матрица | матрица 3 х 3 | ввод матрицы,вывод матрицы,сложение/вычитание двух матриц,умножение двух матриц |
| 20 | Время | час, минута, секунда | вычитание указанного количества секунд,сложение двух моментов времени |
| 21 | Время | час, минута, секунда | прибавление ко времени определенного количества секунд,вычитание из одного момента времени другого |
| 22 | Множество целых чисел | вектор целых чисел | объединение 2 множеств,пересечение 2 множеств,добавление элемента  |

Продолжение таблицы 1

|  |  |  |  |
| --- | --- | --- | --- |
| *1* | *2* | *3* | *4* |
| 23 | Рациональная дробь | числитель, знаменатель | сложение/вычитание 2 дробей,сокращение дроби |
| 24 | Рациональная дробь | числитель, знаменатель | умножение/деление 2 дробей,сравнение 2 дробей |
| 25 | Комплексное число | действительная частьмнимая часть | ввод/вывод комплексного числа,сложение двух комплексных чисел,вычитание двух комплексных чисел. |
| 26 | Комплексное число | действительная частьмнимая часть | ввод/вывод комплексного числа,умножение двух комплексных чисел. |

**Задание 2**. Разработать иерархию классов, описывающих следующие понятия, обосновать свой выбор.

Таблица 2 – Варианты индивидуального задания 2

|  |  |  |
| --- | --- | --- |
| **№ варианта** | **Базовый класс** | **Производные классы** |
| *1* | *2* | *3* |
| 1 | Инструмент | электроинструмент, электродрель |
| 2 | Товар  |  продукты питания, коробка конфет |
| 3 | Устройство хранения информации  |  CD-ROM, CD-RW |
| 4 | Периферийное устройство  |  мышь, беспроводная мышь |
| 5 | Транспортное средство  |  автомобиль, такси |
| 6 | Письменная принадлежность  |  карандаш, цветной карандаш |

Продолжение таблицы 2

|  |  |  |
| --- | --- | --- |
| *1* | *2* | *3* |
| 7 | Точка  |  окружность, закрашенная окружность |
| 8 | Точка  |  линия, пунктирная линия |
| 9 | Растение  |  дерево, лес |
| 10 | Бытовая техника  |  телевизор, стиральная машина |
| 11 | Ювелирное украшение  |  цепочка, кольцо |
| 12 | Устройство ввода  |  мышь, клавиатура |
| 13 | Списки  |  простой, двунаправленный, очередь |
| 14 | Списки  |  простой, двунаправленный, стек |
| 15 | Запись в телефонной книге  |  частное лицо, организация |
| 16 | Пользователь библиотеки  |  студент, преподаватель |
| 17 | Человек  |  студент, преподаватель |
| 18 | Работник  |  работник на окладе, работник с почасовой заработной платой |
| 19 | Конструкция | Строение, жилой дом |

**Лабораторная работа 3. Перегрузка операций и функций**

**Цель и содержание:** знакомство с реализациями принципаполиморфизма в языке С++, приобретение навыка программирования перегруженных функций и операций.

**Теоретическое обоснование**

Полиморфизм является процессом, благодаря которому общий интерфейс применяется к нескольким схожим логически, но технически разным операциям. Подобное отношение может существенно упростить обращение к сложным системам.

Различают два способа определения принадлежности функций‑членов объектам классов: раннее связывание и позднее связывание. Раннее связываение реализуется в С++ через перегруженные операции и функции.

Класс ostream определяет операцию <<, чтобы сделать удобным вывод нескольких объектов одним оператором.

Чтобы определить @, где @ − некоторая операция языка С++, для каждого определяемого пользователем типа вы определяете функцию с именем operator@, которая получает параметры соответствующего типа.

Например:

class ostream {

//...

ostream operator<<(char);

};

ostream ostream::operator<<(char\* p)

{

while {p) buf.sputc{p++);

return \*this;

}

В класс ostream операцию << переопределяет член класса, поэтому s<<p интерпретируется как s.operator<<(p) (если s является объектом класса ostream и p – указатель на символьную переменную). Операция << бинарная, а функция operator<<(char) на первый взгляд имеет только один параметр. Левый операнд бинарной операции передается через стандартный параметр this – указатель на текущий объект класса.

То, что в качестве возвращаемого значения возвращается указатель на объект класса ostream, позволяет применять << к результату операции вывода. Например, s<<p<<q интерпретируется как (s.operator<<(p)).operator<<(q). Так задаются операции вывода для встроенных типов.

С помощью множества операций, заданных как открытые члены класса ostream, вы можете теперь определить оператор << для такого определяемого типа, как complex, не изменяя описание класса ostream:

ostream operator<<(ostream s, complex z)

// у complex две части: действительная real и

// мнимая imag

// печатает complex как (real,imag)

{

return s << "(" << z.real << "," << z.imag << ")';

}

Поскольку operator<<(ostream,complex) не является функцией-членом, для бинарности необходимо два явных параметра. Вывод значений будет производиться в правильном порядке, потому что <<, как и большинство операций С++, компилятор группирует слева направо, то есть a<<b<<c означает (a<<b)<<c. При интерпретации операций компилятору известна разница между функциями-членами и функциями-не членами. Например, если z − комплексная переменная, то s<<z будет расширяться с помощью вызова стандартной функции (не члена) operator<<(s,z).

Как правило, давать разным функциям разные имена − мысль хорошая, но когда некоторые функции выполняют одинаковую работу над объектами разных типов, может быть более удобно дать им одинаковые имена. Использование одного имени для различных действий над различными типами называется **перегрузкой** (overloading). Метод уже используется для основных операций С++: у сложения существует только одно имя, +, но его можно применять для сложения значений целых, плавающих и указательных типов. Эта идея легко расширяется на обработку операций, определенных пользователем, то есть функций. Чтобы уберечь программиста от случайного повторного использования имени, имя может использоваться более чем для одной функции, только если оно сперва описано как перегруженное имя функции. Например:

overload print;

void print(int);

void print(char\*);

Когда вызывается перегруженная функция, компилятор должен понять, к какой из функций с одинаковым именем следует обратиться. Это делается путем *сравнения типов фактических параметров с типами формальных параметров всех функций с перегруженным именем*. Поиск функции, которую надо вызвать, осуществляется за три отдельных шага:

1. Искать функцию соответствующую точно, и использовать ее, если она найдена.
2. Искать соответствующую функцию используя встроенные преобразования и использовать любую найденную функцию.
3. Искать соответствующую функцию используя преобразования, определенные пользователем, и если множество преобразований единственно, использовать найденную функцию.

Например,

overload print(double), print(int);

void f();

{

print(1);

print(1.0);

}

Правило точного соответствия гарантирует, что f( ) напечатает 1 как целое и 1.0 как число с плавающей точкой. Ноль, char или short точно соответствуют параметру. Аналогично, float точно соответствует double.

К параметрам функций с перегруженными именами стандартные С++ правила неявного преобразования типа применяются не полностью. Преобразования, могущие уничтожить информацию, не выполняются. Остаются int в long, int в double, ноль в long, ноль в double и преобразования указателей: преобразование ноль в указатель void\*, и указатель на производный класс в указатель на базовый класс.

Вот пример, в котором преобразование необходимо:

overload print(double), print(long);

void f(int a);

{

print(a);

}

Здесь a может быть напечатано или как double, или как long. Неоднозначность разрешается явным преобразованием типа (или print(long(a)) или print(double(a))).

При этих правилах можно гарантировать, что когда эффективность или точность вычислений для используемых типов существенно различаются, будет использоваться простейший алгоритм (функция). Например,

 overload pow;

 int pow(int, int);

 double pow(double, double); // из <math.h>

 complex pow(double, complex); // из <complex.h>

 complex pow(complex, int);

 complex pow(complex, double);

 complex pow(complex, complex);

**Методика и порядок выполнения лабораторной работы**

1. Изучить теоретический материал по лекциям, дополнительной литературе.
2. Изучить теоретическое обоснование лабораторной работы.
3. Ответить на контрольные вопросы.
4. Выполнить индивидуальное задание.
5. Составить отчет о лабораторной работе.

**Задание к лабораторной работе**

Выполнить перегрузку перечисленных операций, закрепив за ними соответствующий функциональный смысл.

Таблица 3. – Варианты индивидуального задания

|  |  |  |  |
| --- | --- | --- | --- |
| № ва-рианта | Понятие | Символ операции | Значение |
| *1* | *2* | *3* | *4* |
| 1 | Строка символов | [] | доступ к символу строки по его номеру |
| += | добавление символа к концу строки |
| -- | операции удаления последнего символа |
| 2 | Дата в формате (день, месяц, год). | ++ | увеличение даты на 1 сутки |
| -- | уменьшение даты на 1 сутки |
| + | сложение двух дат |
| 3 | Дата в формате (день, месяц, год). | + | прибавление к дате указанного количества дней |
| - | вычитание из даты указанного количества дней |
| % | вычисление числа дней, прошедших между двумя датами |
| 4 | Строка символов | ^ | удаление из строки указанного символа |
| + | сложение 2 строк |
| float | преобразование к типу  |

Продолжение таблицы 3

|  |  |  |  |
| --- | --- | --- | --- |
| *1* | *2* | *3* | *4* |
| 5 | Одномерный массив с указателем на текущий элемент | -- ,++ | перемещение указателя на предыдущий следующий элемент |
| = | присваивание значения текущему элементу |
| \* | умножение значения текущего элемента |
| 6 | Двоичное число | + | сложение двух двоичных чисел |
| \* | умножение двух двоичных чисел |
| - | разность двух двоичных чисел |
| ~ | инвертирование всех битов двоичного числа |
| 7 | Двоичное число | -- | уменьшение двоичного числа на 1 |
| ++ | увеличение двоичного числа на 1 |
| int | преобразование к типу |
| 8 | Восьмеричное число | + | сложение двух восьмеричных чисел |
| \* | умножение двух восьмеричных чисел |
| - | разность двух восьмеричных чисел |
| ^ | возведение в степень |

Продолжение таблицы 3

|  |  |  |  |
| --- | --- | --- | --- |
| *1* | *2* | *3* | *4* |
| 9 | Шестнадцатеричное число | + | сложение двух шестнадцатеричных чисел |
| \* | умножение двух шестнадцатеричных чисел |
| - | разность двух шестнадцатеричных чисел |
| 10 | Вычислительная структура «стек целых чисел» | << | добавление элементов в стек |
| >> | извлечение элемента из стека |
| ! | вычисление суммы элементов, находящихся в стеке |
| 11 | Вычислительная структура «очередь» | << | добавление элементов в очередь |
| >> | извлечение элемента из очереди |
| + | операция добавление в очередь всех элементов другой очереди |
| 12 | Телефонный справочник | Add | добавление записи в книгу |
| ^ | поиск телефона по заданному имени |
| - | удаление записи по заданному имени |
| 13 | Вычислительная структура «множество» | << | добавление элементов во множество |
| >> | извлечение элемента из множества |
| ! | удаление всех элементов из множества |

Продолжение таблицы 3

|  |  |  |  |
| --- | --- | --- | --- |
| *1* | *2* | *3* | *4* |
| 14 | Вычислительная структура «приоритетная очередь» | += | добавление элементов в очередь |
| ~ | извлечение элемента из очереди |
| ^ | удаления всех элементов с приоритетом меньше заданного |
| 15 | Вектор в пятимерном евклидовом пространстве | +/- | сложение/вычитание векторов |
| \* | скалярное произведение векторов |
| ^ | умножение вектора на скаляр |
| 16 | Строка символов | == | сравнение строк |
| - | удаление из строки заданного символа |
| ~ | переворот строки |
| 17 | Прямоугольная матрица | +/- | сложение/вычитание двух матриц |
| \* | умножение двух матриц |
| 18 | Комплексное число, заданное своим модулем и углом | +/- | сложение/вычитание двух комплексных чисел |
| \* | умножение двух комплексных чисел |
| / | деление двух комплексных чисел |
| ^ | возведение в степень |
| 19 | Комплексное число заданное своими вещественной и мнимой частями | +/- | сложение/вычитание двух комплексных чисел |
| \* | умножение двух комплексных чисел |
| / | деление двух комплексных чисел |

Продолжение таблицы 3

|  |  |  |  |
| --- | --- | --- | --- |
| *1* | *2* | *3* | *4* |
| 20 | Время в формате час/минута/секунда | + | сложение момента времени и определенного пользователем количества секунд |
| - | вычитание из одного момента времени другого |
| % | подсчет числа секунд между двумя моментами времени, лежащими в пределах одних суток |
| 21 | Полином | +/- | сложение/вычитание двух полиномов |
| \* | умножение двух полиномов |
| / | деление двух полиномов |
| \* | умножение полинома на число |
| 22 | Множество целых чисел | + | объединение двух множеств |
| & | пересечение двух множеств |
| - | разность двух множеств |
| << | добавление элемента во множество |
| >> | удаление элемента из множества |
| 23 | Рациональная дробь | +/- | сложение/вычитание двух дробей |
| \* | умножение двух дробей |
| / | деление двух дробей |
| ! | приведение дроби к несократимому виду |
| << | вывод дроби на экран |

Продолжение таблицы 3

|  |  |  |  |
| --- | --- | --- | --- |
| *1* | *2* | *3* | *4* |
| 24 | Строка символов с указателем текущего символа | << | вставка строки в текущую позицию |
| ~ | поиск подстроки в строке |
| ^ | замена выделенного блока подстрокой |
| 25 | Список с маркером | << | вставка элемента списка после текущего |
| +/- | вычитание из выделенного элемента списка выделенного элемента другого списка. |
| ^ | возвращение числа совпадающих элементов в двух списках |

**Лабораторная работа 4. Виртуальные функции**

**Цель и содержание:** знакомство с реализациями принципаполиморфизма в языке С++ на стадии выполнения программы, приобретение навыка программирования виртуальных функций.

**Теоретическое обоснование**

Позднее связывание предназначено для работы с динамически создаваемыми объектами. При этом обращение к объекту может выполняться через указатель типа базового класса. Компилятор не может определить на объект какого класса будет указывать указатель, поэтому при исполнении программы должно выполняться определение реального класса объекта и выбор вызываемых методов.

Разница типов связывания заключается в том, что при раннем связывании получаются программы, реализующие максимальное быстродействие за счет небольшой гибкости при исполнении, а при позднем связывании можно достигнуть высокой гибкости программы, проиграв в быстродействии.

Виртуальные функции важны потому, что они позволяют реализовать динамический полиморфизм путем позднего связывания. При определении виртуальной функции в базовом классе перед ее именем должно указываться ключевое слово **virtual**. При этом создается только **прототип** метода, сама функция не описывается. Она должна описываться в производных классах, причем указывать ключевое слово **virtual** не нужно. Важным условием является то, что функции порожденных классов должны полностью соответствовать прототипу, описанному в базовом классе. Различают обычные виртуальные функции, которые *обязательно* должны переопределяться в производных классах и **чистые** виртуальные функции, которые переопределять не обязательно. В отличие от обычной функции, чистая виртуальная функция определяется следующим образом:

virtual возвращаемый\_тип имя\_функции(параметры) = 0;

//Например:

virtual int mylovelyfunc(int) = 0;

 Нельзя создать объект класса, содержащего чисто виртуальную функцию. Но на такой класс можно создать указатель, который в дальнейшем используется для реализации механизма виртуальности.

**Пример выполнения задания**

Реализовать иерархию классов «список: очередь, стек». Функции добавления и удаления элементов сделать виртуальными.

#include <iostream.h>

#include <stdlib.h>

#include <ctype.h>

using namespace std;

//Определение базового класса "список"

class list {

public:

 list \*head; //указатель на начало списка

 list \*tail; //указатель на конец списка

 list \*next; //указатель на следующий элемент списка

 int num; //число, хранящееся в элементе

// списка

list () {head = tail = next = NULL;} //конструктор

 virtual void store(int i) = 0;

 virtual int retrieve() = 0;

};

//Наследование списка "очередь"

class queue: public list {

public:

 void store(int i);

 int retrieve();

};

void queue::store (int i)

{

//функция сохранения числа в элементе очереди

 list \*item;

 item = new queue;

 if (!item) {

 cout << "Ошибка выделения памяти\n";

 exit 1;

 }

 item->num = i;

 //размещение элемента в конце очереди

 if (tail) tail->next = item;

 tail = item;

 item->next = NULL;

 if (!head) head = tail;

}

int queue::retrieve()

{

//функция возвращения элемента очереди

 int i;

 list \*p;

 if (!head) {

 cout << "Список пуст\n";

 return 0;

 }

 //удаление возвращенного элемента очереди

// (из начала очереди)

 i = head->num;

 p = head;

 head = head->next;

 delete p;

 return i;

}

//Наследование списка "стек"

class stack: public list {

public:

 void store(int i);

 int retrieve();

};

void stack::store (int i)

{

//функция сохранения числа в элементе стека

 list \*item;

 item = new stack;

 if (!item) {

 cout << "Ошибка выделения памяти\n";

 exit 1;

 }

 item->num = i;

 //размещение элемента в начале списка

 if (head) item->next = head;

 head = item;

 if (!tail) tail = head;

}

int stack::retrieve()

{

//функция возвращения элемента стека

 int i;

 list \*p;

 if (!head) {

 cout << "Список пуст\n";

 return 0;

 }

 //удаление возвращенного элемента стека

 i = head->num;

 p = head;

 head = head->next;

 delete p;

 return i;

}

int main() {

 list \*p; //указатель типа базового класса

 //работа через указатель с очередью

 queue q\_ob;

 p = &q\_ob;

 p->store(1);

 p->store(2);

 p->store(3);

 cout << "Очередь: ";

 cout << p->retrieve();

 cout << p->retrieve();

 cout << p->retrieve();

 cout << '\n';

//работа через указатель со стеком

 stack s\_ob;

 p = &s\_ob;

 p->store(1);

 p->store(2);

 p->store(3);

 cout << "Стек: ";

 cout << p->retrieve();

 cout << p->retrieve();

 cout << p->retrieve();

 cout << '\n';

 return 0;

}

В данной программе указателю последовательно присваиваются адреса объекта очереди и списка. Несмотря на идентичный интерфейс работы с объектами (имена функций и параметры), в каждом из случаев работа с данными реализуется верно.

**Методика и порядок выполнения лабораторной работы**

1. Изучить теоретический материал по лекциям, дополнительной литературе.
2. Изучить теоретическое обоснование лабораторной работы.
3. Ответить на контрольные вопросы.
4. Выполнить индивидуальное задание.
5. Составить отчет о лабораторной работе.

**Задание к лабораторной работе**

На базе задания лабораторной работы 2 построить иерархию классов, сделав какой-либо из методов класса виртуальным.

**Лабораторная работа 5. Шаблонные функции и классы**

**Цель и содержание:** Приобретение навыков программирования шаблонных классов и функций.

**Теоретическое обоснование**

Шаблоны, которые иногда называют родовыми или параметризованными **типами, позволяют создавать (конструировать) *семейства* родственных** функций и классов.

Шаблон функции имеет вид:

template <список\_параметров\_шаблона> определение\_функции\_или\_класса.

Угловые скобки обязательны.

Шаблон семейства классов определяет способ построения отдельных классов подобно тому, как класс определяет правила построения и формат отдельных объектов. В определении класса, входящего в шаблон, особую роль играет имя класса. Оно является не именем отдельного класса, а параметризованным именем семейства классов.

Определение шаблона может быть только глобальным.

Следуя авторам языка и компилятора C++, рассмотрим векторный класс (в число данных входит одномерный массив). Какой бы тип ни имели элементы массива, в этом классе должны быть определены базовые операции, например, доступ к элементу по индексу. Если тип элементов вектора задавать как параметр шаблона класса, то система будет формировать вектор нужного типа (и соответствующий класс) при каждом определении нового объекта.

//template.vec

Template <class T> //T – параметр шаблона

class Vector

{

T \*data; //Начало одномерного массива

int size; //Количество элементов в массиве

public:

Vector(int); //Конструктор

~Vector(){delete data;} //Деструктор

T& operator[](int i){return data[i];}

};

//Внешнее определение конструктора

template <class T>

Vector <T>::Vector(int i)

{

data = new T[n];

size = n;

}

Когда шаблон введен, у программиста появляется возможность определять конкретные объекты конкретных классов, каждый из которых параметрически порожден от шаблона. Формат определения объекта одного из классов, порождаемых шаблоном классов:

имя\_параметризованного\_класса <фактические\_параметры\_шаблона> имя\_объекта(параметры\_конструктора);

В нашем случае определить вектор, имеющий восемь вещественных координат типа double можно следующим образом:

Vector <double> Z(8)

Пример:

#include "template.vec" //Определение шаблона

// класса

#include <iostream.h>

main()

{

//Создаем целочисленный объект

Vector <int> X(5)

//Создаем символьный объект

Vector <char> C(5)

//Определяем компоненты векторов

for (int i = 0; i < 5; i++

 {X[i] = i; C[i] = 'A' + i;}

for (i = 0; i < 5; i++

 cout << " " << X[i] << ' ' << C[i];

}

**Методика и порядок выполнения лабораторной работы**

1. Изучить теоретический материал по лекциям, дополнительной литературе.
2. Изучить теоретическое обоснование лабораторной работы.
3. Ответить на контрольные вопросы.
4. Выполнить индивидуальное задание 1.
5. Выполнить индивидуальное задание 2.
6. Составить отчет о лабораторной работе.

**Задание к лабораторной работе**

**Задание 1.** Реализовать в виде шаблона класса ниже перечисленные понятия.

Таблица 4 – Варианты индивидуального задания 1

|  |  |
| --- | --- |
| **№ варианта** | **Понятие** |
| 1 | Вычислительная структура «двухмерный массив с контролем выхода индекса за границы массива» |
| 2 | Вычислительная структура «приоритетная очередь» |
| 3 | Вычислительная структура «множество» |
| 4 | Вычислительная структура «словарь» |
| 5 | Вычислительная структура «отсортированный список» |
| 6 | Вычислительная структура «двухмерный разреженный массив» |
| 7 | Вычислительная структура «односвязный список» |
| 8 | Вычислительная структура «двусвязный список» |
| 9 | Вычислительная структура «стек» |
| 10 | Вычислительная структура «очередь» |
| 11 | Вычислительная структура «вагон» |
| 12 | Вычислительная структура «дерево» |

**Задание 2.** Реализовать в виде родовой функции следующие действия.

Таблица 5 – Варианты индивидуального задания 2

|  |  |
| --- | --- |
| **№ варианта** | **Понятие** |
| *1* | *2* |
| 1 | Обмен двух произвольных переменных значениями  |
| 2 | Сортировка массива из N элементов |
| 3 | Возведение числа в производную степень |
| 4 | Вычисление суммы всех элементов массива |
| 5 | Вычисление среднего значения всех элементов массива |

Продолжение таблицы 5

|  |  |
| --- | --- |
| *1* | *2* |
| 6 | Поиск максимального элемента массива |
| 7 | Поиск минимального элемента массива |
| 8 | Вычисление значения выражения A=x2+x7+x9+x/8 |
| 9 | Вычисление значения выражения A=8y3 – lg|x4 - 1| + ln z |
| 10 | Вычисление значения выражения A=(z2 \* x2)/(z \* x2 – y3) |
| 11 | Вычисление значения выражения A=5e2 – 3ln x + 1/lg y |
| 12 | Вычисление значения выражения у=lg|x4-y2|/(xz+35) |
| 13 | Сортировка одномерного массива по возрастанию |

**Лабораторная работа 6. Простейшие классы и объекты**

**Теоретическое введение.**

Классы представляют абстрактные типы данных с открытым интерфейсом и скрытой внутренней реализацией. В классах реализованы базовые принципы *объектноориентированного программирования* (ООП):

1. абстракция данных;
2. инкапсуляция – в классах объединяются данные и методы (функции) для работы с ними, так как лишь через методы возможен доступ к сокрытым данным класса;
3. наследование – в производных классах наследуются члены базового класса;
4. полиморфизм – возможность использования одних и тех же методов для работы с различными объектами базового и порожденных им классов.

Определение простейшего класса без наследования имеет вид:

**class имя\_класса**

**{ // по умолчанию раздел**

**private – частные члены класса**

**public: // открытые функции и переменные класса };**

Имя класса является новым типом данных. Понятию переменной данного типа соответствует понятие объекта класса. Список членов класса включает описание данных и функций. Функции, описания которых находятся в определении класса, называются функциямичленами класса.

Переменные и функции, объявленные в разделе класса по умолчанию или явно как ***private,*** имеют область видимости в пределах класса. Их можно сделать видимыми вне класса, если объявить в разделе ***public:***.Обычно переменные объявляются в разделе ***private***, а функции в разделе ***public***.

Классами в С++ являются также структуры ***(struct*)** и объединения ***(union*).** В отличие от класса члены структуры по умолчанию являются открытыми, а не закрытыми. Кроме того, объединения не могут наследоваться и наследовать.

При реализации функциональной части класса могут быть использованы функции-члены класса, конструкторы, деструкторы, функции-операторы. Функции класса всегда объявляются внутри класса. Определение функции может находиться и внутри класса. Такие функции называются ***inline***-функциями. Обычно определения функций-членов класса помещаются вне класса.

При этом перед именем функции указывается

***имя\_класса::* тип имя\_класса:: имя\_функции (описание аргументов)**

**{ /\*тело функции\*/ }**

Вызов функций осуществляется одним из двух способов:

**имя\_объекта.имя\_функции(аргументы);**

**указатель\_на\_объект -> имя\_функции(аргументы);**

Обращение к данным объекта класса осуществляется с помощью функций, вызываемых из объектов. При этом функции-члену класса передается скрытый указатель ***this*** на объект, вызывающий функцию.

Функции-«друзья» класса, объявляемые в классе со спецификатором ***friend***, указатель ***this*** не содержат. Объекты, с которыми работают такие функции, должны передаваться в качестве их аргументов. Это обычные функции языка С++, которым разрешен доступ к закрытым членам класса.

**Пример.**

**/\*** Создается класс Student. Формируется динамический массив объектов. При тестировании выводится: сформированный список студентов, список студентов заданного факультета, список студентов для заданных факультета и курса.\*/

#include <conio.h>

#include <string.h> #include <iostream.h> struct date // дата рождения {char daymon[6];

 int year; };

//======= class Student ================= class Student{ char name[30]; //private date t; char adr[30], fac[20];

 int kurs; public: Student(); char \*getfac(); int getkurs();

 void show();

 };

Student::Student()

 {cout<<"Input name:"; cin>>name; cout<<"Input date of born\n"; cout<<"Day.mon:"; cin>>t.daymon; cout<<"Year:"; cin>>t.year; cout<<"Input adr:"; cin>>adr; cout<<"Input fac:"; cin>>fac; cout<<"Input kurs:"; cin>>kurs;

}

void Student::show()

 {

 cout<<"Name :"<<name<<endl; cout<<"Was born :"<<t.daymon<<'.'<<t.year<<endl; cout<<"Address :"<<adr<<endl; cout<<"Fac :"<<fac<<endl;

 cout<<"Kurs :"<<kurs<<endl;

 }

char \*Student::getfac() { return fac; } int Student::getkurs() { return kurs; } void spisfac(Student spis[],int n)//список студентов заданного факультетата

 {char fac[20];

 cout<<"Input faculty:"; cin>>fac;

 for(int i=0;i<n;i++) if(strcmp(spis[i].getfac(),fac)==0)spis[i].show();

 }

void spisfackurs(Student spis[],int n)

//список студентов заданных факультета и курса

 {int i,k; char fac[20]; cout<<"Input faculty:"; cin>>fac; cout<<"Input the course:"; cin>>k;

 for(i=0;i<n;i++) if ((strcmp(spis[i].getfac(),fac)==0)&&(spis[i].getkurs()==k)) spis[i].show();

 }

//========= main ================

 void main() { Student \*spis; int n;

 cout<<"Input a number of students: "; cin>>n; spis=new Student [n];

 for(int i=0;i<n;i++) {

 cout<<"=============================="<<endl; spis[i].show();

 }

spisfac(spis,n); spisfackurs(spis,n); delete [] spis; cout<<"press any key!" while(!kbhit());

}

**Задания для самостоятельного решения**

Разработать классы для описанных ниже объектов. Включить в класс методы set (…), get (…), show (…). Определить другие методы.

1. **Student**: Фамилия, Имя, Отчество, Дата рождения, Адрес, Телефон, Факультет, Курс. Создать массив объектов. Вывести:

а) список студентов заданного факультета;

б) списки студентов для каждого факультета и курса;

в) список студентов, родившихся после заданного года.

1. **Abiturient**: Фамилия, Имя, Отчество, Адрес, Оценки. Создать массив объектов. Вывести:

а) список абитуриентов, имеющих неудовлетворительные оценки;

б) список абитуриентов, сумма баллов у которых не меньше заданной;

в) выбрать N абитуриентов, имеющих самую высокую сумму баллов, и список абитуриентов, имеющих полупроходной балл.

1. **Aeroflot**: Пункт назначения, Номер рейса, Тип самолета, Время вылета, Дни недели. Создать массив объектов. Вывести:

а) список рейсов для заданного пункта назначения;

б) список рейсов для заданного дня недели;

в) список рейсов для заданного дня недели, время вылета для которых больше заданного.

1. **Book**: Автор, Название, Издательство, Год, Количество страниц.

Создать массив объектов. Вывести:

а) список книг заданного автора;

б) список книг, выпущенных заданным издательством;

в) список книг, выпущенных после заданного года.

1. **Worker**: Фамилия и инициалы, Должность, Год поступления на работу, Зарплата. Создать массив объектов. Вывести:

а) список работников, стаж работы которых на данном предприятии превышает заданное число лет;

б) список работников, зарплата которых больше заданной;

в) список работников, занимающих заданную должность.

1. **Train**: Пункт назначения, Номер поезда, Время отправления, Число общих мест, Купейных, Плацкартных. Создать массив объектов. Вывести:

а) список поездов, следующих до заданного пункта назначения;

б) список поездов, следующих до заданного пункта назначения и отправляющихся после заданного часа;

в) список поездов, отправляющихся до заданного пункта назначения и имеющих общие места.

1. **Product**: Наименование, Производитель, Цена, Срок хранения, Количество. Создать массив объектов. Вывести:

а) список товаров для заданного наименования;

б) список товаров для заданного наименования, цена которых не превышает указанной;

в) список товаров, срок хранения которых больше заданного.

1. **Patient**: Фамилия, Имя, Отчество, Адрес, Номер медицинской карты, Диагноз. Создать массив объектов. Вывести:

а) список пациентов, имеющих данный диагноз;

б) список пациентов, номер медицинской карты которых находится в заданном интервале.

1. **Bus**: Фамилия и инициалы водителя, Номер автобуса, Номер маршрута, Марка, Год начала эксплуатации, Пробег. Создать массив объектов. Вывести:

а) список автобусов для заданного номера маршрута;

б) список автобусов, которые эксплуатируются больше 10 лет;

в) список автобусов, пробег у которых больше 10 000 км.

1. **Customer**: Фамилия, Имя, Отчество, Адрес, Телефон, Номер кредитной карточки, Номер банковского счета. Создать массив объектов. Вывести:

а) список покупателей в алфавитном порядке;

б) список покупателей, номер кредитной карточки которых находится в заданном интервале.

1. **File**: Имя файла, Размер, Дата создания, Количество обращений. Создать массив объектов. Вывести:

а) список файлов, упорядоченный в алфавитном порядке;

б) список файлов, размер которых превышает заданный;

в) список файлов, число обращений к которым превышает заданное.

1. **Word**: Слово, Номера страниц, на которых слово встречается (от 1 до 10), Число страниц. Создать массив объектов. Вывести:

а) слова, которые встречаются более чем на N страницах;

б) слова в алфавитном порядке;

в) для заданного слова номера страниц, на которых оно встречается.

1. **House**: Адрес, Этаж, Количество комнат, Площадь. Создать массив объектов. Вывести:

а) список квартир, имеющих заданное число комнат;

б) список квартир, имеющих заданное число комнат и расположенных на этаже, который находится в определенном промежутке;

в) список квартир, имеющих площадь, превосходящую заданную.

1. **Phone**: Фамилия, Имя, Отчество, Адрес, Номер, Время внутригородских разговоров, Время междугородних разговоров. Создать массив объектов. Вывести:

а) сведения об абонентах, время внутригородских разговоров которых превышает заданное;

б) сведения об абонентах, воспользовавшихся междугородней связью;

в) сведения об абонентах, выведенные в алфавитном порядке.

1. **Person**: Фамилия, Имя, Отчество, Адрес, Пол, Образование, Год рождения. Создать массив объектов. Вывести:

а) список граждан, возраст которых превышает заданный;

б) список граждан с высшим образованием;

в) список граждан мужского пола.

**Лабораторная работа 7. Разработка классов С++**

**Теоретическое введение.**

При разработке класса необходимо определить данные класса и его методы, конструкторы и деструкторы.Конструктор – это функция-член класса, которая вызывается автоматически при создании статического или динамического объекта класса. Он инициализирует объект и переменные класса. У конструктора нет возвращаемого значения, но он может иметь аргументы и быть перегружаемым.

Противоположные конструктору действия выполняет деструктор, который вызывается автоматически при уничтожении объекта. Деструктор имеет то же имя, что и класс, но перед ним стоит ‘~’. Деструктор можно вызывать явно в отличие от конструктора. Конструкторы и деструкторы не наследуются, хотя производный класс может вызывать конструктор базового класса.

*Операторы-функции.* Используются для введения операций над объектами, связываемых с символами:

***+ , - , \* , / , % , ^ , & , | , ~ , ! , = , < , > ,+= , [] , -> , ( ) , new, delete***.

Оператор-функция является членом класса или дружественной (***friend***) классу. Общая форма оператор-функции-члена класса:

***возвращаемый\_тип***

***имя\_класса*::operator#(список\_аргум) {/\*тело функции\*/}**

После этого вместо ***operator#(a,b)*** можно писать ***a#b***. Здесь # представляет один из введенных выше символов. Примерами являются операторы ***>>*** и ***<<*** – перегружаемые операторы вводавывода. Отметим, что при перегрузке нельзя менять приоритет операторов и число операндов. Если оператор-функция-член класса перегружает бинарный оператор, то у функции будет только один параметр-объект, стоящий справа от знака оператора. Объект слева вызывает оператор-функцию и передается неявно с помощью указателя ***this***. В дружественную функцию указатель ***this*** не передается, поэтому унарный оператор имеет один параметр, а бинарный – два.

Оператор присваивания не может быть дружественной функцией, а только членом класса.

**Пример.** Создается класс Polynom. В головной программе выполняется тестирование класса.

#include <iostream.h>

#include <conio.h> #include <math.h>

class Polynom {

 int n; double \*koef; public:

 Polynom(); //конструкторы

 Polynom(int k);

 Polynom(int k,double \*mas);

 Polynom(const Polynom&ob); //конструктор копирования

 ~Polynom(){delete[]koef;} void GiveMemory(int k); void SetPolynom(int k,double \*mas); void SetDegree(int k){n=k;}; //установить степень void CalculateValue(double x); //вычислить значение int GetDegree(){return n;}; //получить степень double GetOneCoefficient(int i){return(koef[i]);};

 Polynom operator+(Polynom ob); //перегрузка операторов Polynom operator\*(Polynom ob);

 double& operator[](int i){return(koef[i]);}//перегрузка []

 Polynom& operator = (const Polynom p) {

 if(&p==this) return \*this; if(koef) delete [] koef; n=p.n;

 koef=new double [p.n+1]; for(int i=0;i<=p.n;i++) koef[i]=p.koef[i]; return \*this;

}

 friend ostream& operator<<(ostream& mystream,Polynom &ob); friend istream& operator>>(istream& mystream,Polynom &ob); int min(int n,int m) {return (n<m)? n:m; } int max(int n,int m)

 {return (n>m)? n:m; }

};

//\*\*\*\*\*\*\*\*\*\*\* Polynom() \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Polynom::Polynom() { randomize(); n=random(5); koef=new double[n+1]; if(!koef){cout<<"Error";getch();return;}

 for(int i=n;i>=0;i--) koef[i]=random(10)-5;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\* Polynom(int k) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Polynom::Polynom(int k)

{ n=k; koef=new double[n+1]; if(!koef){cout<<"Error";getch();return;}

 for(int i=n;i>=0;i--) koef[i]=random(10)-5;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Polynom(int k,double mas[]) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Polynom::Polynom(int k,double mas[])

{n=k;

 koef=new double[n+1]; if(!koef){cout<<"Error";getch();return;}

 for(int i=n;i>=0;i--) koef[i]=mas[i];

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Polynom(const Polynom&ob) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Polynom::Polynom(const Polynom&ob)

{n=ob.n; koef=new double[n+1]; if(!koef){cout<<"Error";getch();return;} for(int i=0;i<=n;i++) koef[i]=ob.koef[i];

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* void GiveMemory(int k) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* void Polynom::GiveMemory(int k)

{ if(koef) delete [] koef; koef=new double[k+1];

 if(!koef){cout<<"Error";getch();return;}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SetPolynom \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Polynom::SetPolynom(int k,double \*mas)

{ n=k; if(koef) delete [] koef; koef = new double [n+1]; for(int i=n;i>=0;i--) koef[i]=mas[i];

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CalculateValue \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Polynom::CalculateValue(double x=1.0)

{ double s;

 int i;

 for(s=koef[0],i=1;i<=n;i++) s=s+koef[i]\*pow(x,i);

 cout<<"f("<<x<<")="; cout<<s<<endl;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Polynom operator+(Polynom ob) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Polynom Polynom::operator+(Polynom ob)

{ int i;

 Polynom rab; rab.GiveMemory(max(n,ob.GetDegree())); for(i=0;i<=min(n,ob.GetDegree());i++) rab.koef[i]=koef[i]+ob.GetOneCoefficient(i); if(n<ob.GetDegree())

 {

 for(i=min(n,ob.GetDegree())+1;i<=ob.GetDegree();i++)

 rab.koef[i]=ob.GetOneCoefficient(i); rab.n=ob.GetDegree();

 } else

 {

 for(i=min(n,ob.GetDegree())+1;i<=n;i++) rab.koef[i]=koef[i]; rab.n=n;

 } return rab;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Polynom operator\*(Polynom ob) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Polynom Polynom::operator\*(Polynom ob)

{

 int i,j,k; double s; Polynom rab; rab.GiveMemory(n+ob.GetDegree());

 for(i=0;i<=n+ob.GetDegree();i++)

 { s=0; for(j=0;j<=n;j++)

 for(k=0;k<=ob.GetDegree();k++) if(j+k==i)s=s+koef[j]\*ob.GetOneCoefficient(k); rab.koef[i]=s;

 }

 rab.n=n+ob.GetDegree();

 return rab;

}

//\*\*\*\*\*\*\*\*\*\* ostream& operator<<(ostream& mystream,Polynom &ob) \*\*\*\*\*\* ostream& operator<<(ostream& mystream,Polynom &ob) { char c=' '; //пропустим “+” перед первым коэффициентом for(int i=ob.n;i>=0;i--) { double ai=ob.koef[i]; if(ai==0) continue; else {if(ai>0) mystream<<c; mystream<<ai;} if(i==0) continue; else mystream<<"x"; if(i==1) continue; else mystream<<"^"<<i; if(ai!=0)c='+';

 }

 if(c==' ')mystream<<0; mystream<<endl; return mystream;

}

//\*\*\*\*\*\*\*\*\* istream& operator>>(istream& mystream,Polynom &ob) \* istream& operator>>(istream& mystream,Polynom &ob)

{

 int i;

 cout<<"Enter Degree:"; mystream>>ob.n; cout<<endl; for(i=ob.n;i>=0;i--)

 {

 cout<<"Enter koeff "<<i<<":"; mystream>>ob.koef[i];

 }

 return mystream;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MAIN \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* int main(int argc, char\* argv[])

{ const int m=3;

 Polynom f,g,masp[m],\*p1,s; int n=5,i; double K[6]={1.0,3.2,0.0,4.1,0.0,1.1};

 p1=new Polynom(n,K); cout<<\*p1; p1->CalculateValue(2.0); cin>>f; cout<<" f(x)= "; cout<<f; cout<<" g(x)= "; cout<<g; s=f+g; cout<<"f(x)+g(x) = "; cout<<s; s=f\*g; cout<<" f(x)\*g(x) = "; cout<<s; s=masp[0]; cout<<masp[0]; for(i=1;i<m;i++)

 { s=s+masp[i]; cout<<masp[i];} cout<<"Summa: "; cout<< s;

 while(!kbhit()); delete p1; return 0;

 }

**Задания для самостоятельного решения**

Разработать перечисленные ниже классы. При разработке каждого класса возможны два варианта решения: а) данные-члены класса представляют собой переменные и массивы фиксированной размерности; б) память для данных-членов класса выделяется динамически.

1. «**Комплексное число**» – **Complex**. Класс должен содержать несколько конструкторов и операции для сложения, вычитания, умножения, деления, присваивания. Создать два вектора размерности *n* из комплексных координат. Передать их в функцию, которая выполняет сложение комплексных векторов.
2. Определить класс «**Дробь**» – **Fraction** в виде пары (*m n*, ). Класс должен содержать несколько конструкторов. Реализовать методы для сложения, вычитания, умножения и деления дробей. Перегрузить операции сложения, вычитания, умножения, деления, присваивания и операции отношения. Создать массив объектов и передать его в функцию, которая изменяет каждый элемент массива с четным индексом путем добавления следующего за ним элемента массива.
3. Разработать класс «**Вектор**» – **Vector** размерности *n*. Определить несколько конструкторов, в том числе конструктор копирования. Реализовать методы для вычисления модуля вектора, скалярного произведения, сложения, вычитания, умножения на константу. Перегрузить операции сложения, вычитания, умножения, инкремента, декремента, индексирования, присваивания для данного класса. Создать массив объектов. Написать функцию, которая для заданной пары векторов будет определять, являются ли они коллинеарными или ортогональными.
4. Определить класс «**Квадратная матрица**» – **Matrix**. Класс должен содержать несколько конструкторов, в том числе конструктор копирования. Реализовать методы для сложения, вычитания, умножения матриц; вычисления нормы матрицы. Перегрузить операции сложения, вычитания, умножения и присваивания для данного класса. Создать массив объектов класса **Matrix** и передать его в функцию, которая изменяет *i* -ю матрицу путем возведения ее в квадрат. В головной программе вывести результат.
5. Разработать класс **«Многочлен» – Polynom** степени *n*. Написать несколько конструкторов, в том числе конструктор копирования. Реализовать методы для вычисления значения полинома; сложения, вычитания и умножения полиномов. Перегрузить операции сложения, вычитания, умножения, инкремента, декремента, индексирования, присваивания. Создать массив объектов класса. Передать его в функцию, вычисляющую сумму полиномов массива и возвращающую полином-результат, который выводится на экран в головной программе.
6. Определить класс **«Стек» – Stack**. Элементы стека хранятся в массиве. Если массив имеет фиксированную размерность, то предусмотреть контроль выхода за пределы массива. Если память выделяется динамически и ее не хватает, то увеличить размер выделенной памяти. Включение элементов в стек и их извлечение реализовать как в виде методов, так и с помощью перегруженных операций. Создать массив объектов класса **Stack**. Передавать объекты в функцию, которая удаляет из стека первый (сверху), третий, пятый и т. д. элементы.
7. Построить классы для описания плоских фигур: круг, квадрат, прямоугольник. Включить методы для изменения объектов, перемещения на плоскости, вращения. Перегрузить операции, реализующие те же действия. Выполнить тестирование класса, создав массив объектов.
8. Определить класс «**Строка**» – **String** длины *n*. Написать несколько конструкторов, в том числе конструктор копирования. Реализовать методы для выполнения конкатенации строк, извлечения символа из заданной позиции, сравнения строк. Перегрузить операции сложения, индексирования, отношения, добавления (+=), присваивания для данного класса. Создать массив объектов и передать его в функцию, которая выполняет сортировку строк.
9. Разработать класс «**Множество (целых чисел, символов, строк** и т. д.)» – **Set** мощности *n*. Написать несколько конструкторов, в том числе конструктор копирования. Реализовать методы для определения принадлежности заданного элемента множеству, пересечения, объединения, разности двух множеств. Перегрузить операции сложения, вычитания, умножения (пересечения), индексирования, присваивания. Создать массив объектов и передавать пары объектов в функцию, которая строит множество, состоящее из элементов, входящих только в одно из заданных множеств, т. е. (*A*∪ *B*) \ (*A*∩ *B*), и возвращает его в головную программу.
10. Разработать класс для массива строк. Написать несколько конструкторов, в том числе конструктор копирования. Реализовать методы для поэлементной конкатенации двух массивов, упорядочения строк в лексикографическом порядке, слияния двух массивов с удалением повторяющихся строк, а также для вывода на экран всего массива и заданной строки. Перегрузить операции сложения, умножения, индексирования, присваивания для данного класса. Создать массив объектов и передавать объекты в функцию, которая выполняет слияние объектов и для полученного объекта-результата производит лексикографическое упорядочения строк.
11. Составить описания класса, обеспечивающего представление матрицы заданного размера *n*×*m* и любого минора в ней. Память для матрицы выделять динамически. Написать несколько конструкторов, в том числе конструктор копирования. Реализовать методы для отображения на экране как матрицы в целом, так и заданного минора, а также для изменения минора; сложения, вычитания, умножения миноров. Перегрузить операции сложения, вычитания, умножения и присваивания для данного класса. Создать массив объектов данного класса и передать его в функцию, которая изменяет для *i* -й матрицы ее минор путем умножения на константу.
12. Построить класс «**Булев вектор»** – **BoolVector** размерности *n*. Определить несколько конструкторов, в том числе конструктор копирования. Реализовать методы для выполнения поразрядных конъюнкции, дизъюнкции и отрицания векторов, а также подсчета числа единиц и нулей в векторе. Реализовать те же действия над векторами с помощью перегруженных операций. Перегрузить операции отношения и присваивания для данного класса. Создать массив объектов. Передавать объекты в функцию, которая будет их изменять по формуле

*A*= *A*∨ *B*.

1. Реализовать класс «**Троичный вектор»** – **Tvector** размерности *n*. Компоненты вектора принимают значения из множества {0,1, *X*}.

Два троичных вектора *tk*=(*t*1*k*,…,t*nk*) и *tl*=(*t*1*l*,…,*tnl*) называются ортогональными, если существует такое *i*, что *tik*, *til*∈{0,1} и *tik*≠*til*. Операция пересечения не ортогональных векторов выполняется покомпонентно по следующим правилам: 1∩1=1∩*X*=*X*∩1=1, 0∩0=0∩*X*=*X*∩0=0, *X*∩*X*=*X*. Реализовать методы для проверки векторов на ортогональность, для пересечения не ортогональных векторов, сравнения векторов, подсчета числа компонент, равных *X*. Осуществить те же действия над векторами с помощью перегруженных операций. Перегрузить операцию присваивания для данного класса. Выполнить тестирование класса, создав массив объектов.

**Лабораторная работа 8. Классы для работы с динамическими структурами данных**

**Теоретическое введение**.

Объекты классов могут храниться в виде массивов или динамических связных списков. Если класс содержит конструктор, массив может быть инициализирован, причем конструктор вызывается столько раз, сколько задается элементов массива:

**samp ob[4]={1,2,3,4};**

Список инициализации – это сокращение общей конструкции:

**samp ob[4]={samp(1,2),samp(3,4),samp(5,6),samp(7,8)};**

При создании динамических объектов используется оператор ***new***, который вызывает конструктор и производит инициализацию. Для разрушения динамического объекта используется оператор ***delete***, который может помещаться в деструкторе.

Кроме указателей на классы используются ссылки. Ссылка является скрытым указателем и работает как другое имя переменной. При передаче объекта через ссылку в функцию передается адрес объекта и не делается его копия. Это уменьшает вероятность ошибок, связанных с выделением динамической памяти и вызовом деструктора. Так, при передаче в функцию параметра-объекта может возникнуть ошибка из-за разрушения деструктором на выходе копии объекта, которая должна быть исправлена созданием конструктора копирования. В такой ситуации лучше передать в функцию ссылку на объект и возвратить ссылку на объект.

**Пример.** Создается класс **Tree** (бинарное дерево). Информационная часть узла дерева содержит целое число. Тестирование класса выполняется с помощью меню, которое позволяет сформировать дерево, вывести содержимое его узлов в порядке возрастания, найти узел по ключу, вывести содержимое листьев дерева (вершин, не имеющих потомков).

#include"vip\menu.cpp" //реализация работы с меню

#include <conio.h>

#include <string.h> #include <iostream.h> char bufRus[256]; char\*Rus(const char\*text){ CharToOem(text,bufRus);

 return bufRus;}

 struct node

 {

 int n; //информационное поле узла дерева

 int count;

 node\*left,\*right;

 };

 class Tree

 { public: node\*root;

 Tree(){root=0;}

 Tree(int t); // Формирование дерева из t случайных чисел void CopyTree(node\*&rootnew,node\*rootold);

 /\* Копирует дерево с корнем rootold в дерево с корнем rootnew. В результате деревья находятся в различных динамических участках памяти.\*/

 Tree(const Tree&ob); //конструктор копирования

// Рекурсивная функция, используемая в деструкторе (освобождение памяти) void DelTree(node \*wer); ~Tree(){DelTree(root);} void Push(node\*&wer,int data);// Вставка элемента в дерево void Look(node\*wer); //- Вывод дерева на экран node\*Find(node\*wer,int key); // Поиск по ключу

 void PrintLeaves(node \*wer); // Вывод листьев дерева на экран

 };

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Tree::Tree(int t) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Tree::Tree(int t)

{ root=0;

 for(int i=0;i<t;i++)

 Push(root,random(10)-5);

}

void Tree::CopyTree(node\*&rootnew,node\*rootold)

{

 if(rootold->left!=0)

 {Push(rootnew,(rootold->left)->n);CopyTree(rootnew,rootold->left);} if(rootold->right!=0)

 {Push(rootnew,(rootold->right)->n);CopyTree(rootnew,rootold->right);} }

Tree::Tree(const Tree&ob)

{

 if(ob.root==0)root=0; else { root=new node; root->n=ob.root->n; root->count=1; root->left=0; root->right=0;

 CopyTree(root,ob.root);

 }

}

void Tree::DelTree(node \*wer)

{ if(wer->left!=0)DelTree(wer->left); if(wer->right!=0)DelTree(wer->right);

 delete wer; }

void Tree::Push(node\*&wer,int data)

 {

 if(wer==0)

 {

 wer=new node; wer->n=data; wer->left=0;wer->right=0;

 wer->count=1;

 }

 else if(data<wer->n)Push(wer->left,data); else if(data>wer->n)Push(wer->right,data);

 else wer->count++;

 }

void Tree::Look(node\*wer)

{ if(wer!=0)

 {

 Look(wer->left);

 cout<<Rus("Число: ")<<wer->n<<" - "<<wer->count;

 cout<<Rus(" штук")<<endl;

 Look(wer->right);

 } }

node\* Tree::Find(node\*wer,int key)

{ if(wer==0) return 0; else if(key<wer->n) return Find(wer->left,key); else if(key>wer->n) return Find(wer->right,key);

 else return wer;

}

void Tree::PrintLeaves(node \*wer)

{ if(wer==0)return; else if( (wer->left==0)&&(wer->right==0) ) { cout<<Rus(“Число: “)<<wer->n<<”-“<<wer->count; cout<<Rus(“штук”)<<endl;

 } else

 {

 PrintLeaves(wer->left);

 PrintLeaves(wer->right);

 }

}

//-------------------------------- MAIN ---------------------------------------- int main(int argc, char\* argv[])

{

 Tree tr; node \*u; int k=0,max,kol; char menu[][100]={ {" PushElement "}, {" ShowTree "}, {" FindElement "},

 {" PrintLeaves "}, {" EXIT "}, };

 kol=5;//КОЛИЧЕСТВО СТРОК МЕНЮ. Используется в выравнивании строк

 // меню по центру.

//------------------ВЫРАВНИВАНИЕ СТРОК МЕНЮ ПО ЦЕНТРУ---------------- max=viravnivaniestrok(menu,kol);

//------------------------ МЕНЮ НА ЭКРАНЕ--------------------------------------- textmode(C80);

 while(1){ switch(mmm(kol,menu,max,k))

 { case 0: {

 int data;

 cout<<Rus("Введите число:");

 cin>>data; tr.Push(tr.root,data); k=0;break;

 } case 1: { if(tr.root==0)cout<<Rus("Дерево пустое"); else

 {

 cout<<Rus("Наше дерево:")<<endl; tr.Look(tr.root);

 } while(!kbhit()); k=1;break;

 } case 2: { if(tr.root==0)cout<<Rus("Дерево пустое"); else { int key;

 cout<<Rus("Введите искомое число:"); cin>>key; if((u=tr.Find(tr.root,key))!=0){ cout<<Rus("Элементов: "); cout<<key;

 cout<<Rus(" найдено "); cout<<u->count<<Rus(" штук"); }

 else cout<<Rus("Таких элементов нет!");

 }

 while(!kbhit()); k=2;break;

 } case 3: { if(tr.root==0)cout<<Rus("Дерево пустое"); else{ cout<<Rus("Листья:")<<endl; tr.PrintLeaves(tr.root);

 }

 while(!kbhit()); k=3;break;

 } case 4:{ exit(0);

 } } } return 0;

}

**Задания для самостоятельного решения**

При решении задач необходимо описать класс, который используется для представления элементов динамической структуры данных. Затем разрабатывается класс для работы с используемой динамической структурой данных, которая при тестировании класса может быть построена путем ввода данных: a) с клавиатуры; б) из файла.

Возможны два варианта решения:

а) динамическая структура данных постоянно хранится в памяти;

б) динамическая структура данных хранится в файле.

1. Создать класс для работы со стеком. Элемент стека – действительное число. Применить класс для вывода возрастающих серий последовательности действительных чисел: a) в обратном порядке; б) в том же порядке (серия – упорядоченная последовательность максимальной длины).
2. Построить класс для работы со стеком. Элемент стека – целое число. Ввести две неубывающие последовательности чисел в два стека. Использовать третий стек для слияния двух последовательностей в одну неубывающую.
3. Создать класс для работы со стеком. Элемент стека – символ. Сформировать два стека, содержащие последовательности символов. Подсчитать общее число элементов в стеках, предусмотреть восстановление их исходного расположения.
4. Создать класс для работы со стеком. Элемент стека – символ. Использовать стек для проверки правильности расстановки скобок трех типов (круглых, квадратных и фигурных) в выражении.
5. Построить класс для работы с односвязным списком. Элемент списка – действительное число. Сформировать список, содержащий неубывающую последовательность чисел, и преобразовать его так, чтобы последовательность была невозрастающей. Для этого необходимо совершить переворот списка, т. е. такую переустановку указателей в списке, при которой элементы его следуют друг за другом в обратном порядке.
6. Построить класс для работы с односвязным списком. Элементы списка – целые числа. Сформировать список, упорядочить элементы списка по возрастанию, используя сортировку: a) методом выбора; б) методом пузырька; в) методом вставки.
7. Построить класс для работы с односвязным списком. Элементы списка – действительные числа. Создать два упорядоченных по невозрастанию списка, слить их в один (также упорядоченный по невозрастанию), построив новый список.
8. Построить класс для работы с односвязным списком. Элементы списка – слова. Создать список, содержащий некоторую последовательность слов. Заменить в списке каждое вхождение заданного слова другим (также заданным).
9. Построить класс для работы с односвязным списком. Создать два списка: List1 и List2. Проверить, содержатся ли элементы списка List1 в списке List2 в указанном списком List1 порядке.
10. Построить класс для работы с односвязным списком. Элементы списка – целые числа. Создать список List1. Построить список List2, содержащий порядковые номера максимальных элементов списка List1.
11. Построить класс для работы с двусвязным списком. Элементы списка – действительные числа. Создать список List1, содержащий последовательность *x*1, *x*2, ... , *xn* . Построить список List2, содержащий последовательность *x*1, *x x xn*, 2, *n*−1,..., *x xn*, 1.
12. Создать класс для работы с бинарным деревом, узлы которого содержат целые числа. Построить дерево, затем копию дерева. Подсчитать число листьев в нем (листьями называются узлы, не содержащие поддеревьев).
13. Построить класс для работы с бинарным деревом, узлы которого содержат действительные числа. Создать дерево. Определить высоту дерева (максимальное число узлов, принадлежащих пути от корня дерева до любого из его листьев). Подсчитать число элементов, равных максимальному.
14. Построить класс для работы с бинарным деревом, узлы которого содержат действительные числа. Создать дерево для заданной последовательности чисел. Используя его, упорядочить последовательность по возрастанию, убыванию.
15. Построить класс для работы со списком. Элемент списка содержит информацию о заявке на авиабилет: пункт назначения, номер рейса, фамилию и инициалы пассажира, желаемую дату вылета.

Программа должна обеспечивать: хранение всех заявок в виде списка, добавление заявок в список, удаление заявок, вывод заявок по заданному номеру рейса и дате вылета, вывод всех заявок.

1. Построить класс для работы с бинарным деревом, узел которого содержит информацию о заявках на авиабилеты (в последовательности, используемой для упорядочения заявок): желаемую дату вылета, номер рейса, фамилию и инициалы пассажира, пункт назначения.

Программа должна обеспечивать: хранение всех заявок в виде бинарного дерева, добавление заявок, удаление заявок, вывод заявок по заданному номеру рейса и дате вылета, вывод всех заявок.

1. Построить класс для работы со списком, который содержит динамическую информацию о наличии автобусов в парке: номер автобуса, фамилию и инициалы водителя, номер маршрута, признак местонахождения автобуса – на маршруте или в парке.

Программа должна обеспечивать: начальное формирование списка, введение номера автобуса при выезде и установление программой значения признака «автобус на маршруте». Аналогичным образом изменяется информация об автобусе при его возвращении в парк. По запросу выдаются сведения об автобусах, находящихся в парке, или об автобусах, находящихся на маршруте.

1. Решить предыдущую задачу, используя не список, а бинарное дерево.
2. Построить класс для работы с бинарным деревом, содержащим англо-русский словарь.
3. Построить класс для работы со списком, содержащим информацию о поездах дальнего следования. Элемент списка содержит следующую информацию о поезде: номер поезда, станция назначения, время отправления.

Составить программу, которая обеспечивает первоначальное формирование списка, производит вывод списка, вводит номер поезда и выводит информацию о нем, вводит название станции назначения и выводит данные о всех поездах, следующих до этой станции.

**Лабораторная работа 9. Шаблоны классов С++**

**Теоретическое введение**.

С помощью шаблонов можно создавать родовые ***(generic)*** функции и классы. Родовая функция определяет базовый набор операций, которые будут применяться к разным типам данных, получаемых функцией в качестве параметра.

Определение функции с ключевым словом ***template*** (шаблон) имеет вид:

**template<class Ttype>тип имя\_функции(список аргументов)**

**{//тело функции}**

Здесь ***Ttype*** – фиктивный тип, который используется при объявлении аргументов и локальных переменных функции. Компилятор заменит этот фиктивный тип на один из реальных и создаст соответственно несколько перегружаемых функций. При этом перегружаемые функции являются ограниченными, поскольку выполняют одни и те же действия над данными различных типов.

С помощью шаблона класса можно создать класс, реализующий стек, очередь, дерево и т. д. для любых типов данных. Компилятор будет генерировать правильный тип объекта на основе типа, задаваемого при создании объекта. Общая форма объявления шаблона класса:

**template <class Ttype> class имя\_класса**

**{ //имена класса, поля класса**

**}**

Здесь ***Ttype*** – фиктивное имя типа, которое заменится компилятором на фактическое. Шаблон класса может иметь больше одного родового типа данных:

**template<class Type1,class Type2> class m {Type1 a;Type2 b;}**

**Пример.** Создается шаблон класса для работы с односвязным списком. Тестирование класса выполняется с использованием меню, которое позволяет создать список из чисел типов integer, float, double и выполнить типичные операции с ним.

#include <conio.h>

#include <string.h>

#include <iostream.h> #include"vip\menu.cpp" void ListForInt(); void ListForFloat(); void ListForDouble(); char bufRus[256]; char\*Rus(const char\*text){ CharToOem(text,bufRus);

 return bufRus;

}

//шаблон класса для работы с односвязным списком template<class mytype>class List {

//внутренний класс, для представления элементов списка class Node{ public:

 mytype d;

 Node\* next;

 Node(mytype dat=0){d=dat; next=0;}

 };

 Node\* pbeg; //указатель на начало списка public:

 List(){pbeg=0;} //конструктор

 ~List(); //деструктор

 Node \* Add(mytype d); //добавление в конец списка

 Node \* Find(mytype key); //поиск по ключу

 Node \* Insert(mytype key,mytype d); //вставка узла d после узла с

 // ключом key bool Remove(mytype key); //удаление узла

 void Print(); //печать списка

 };

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*~List() \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //ДЕСТРУКТОР. Освобождает память для всех узлов списка

template<class mytype> List<mytype>::~List(){

 if(pbeg!=0){ Node\* pv=pbeg; while(pv){ pv=pv->next; delete pbeg;

 pbeg=pv;

 }

 }

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* void Add(mytype d) \*\*\*\*\*\*\*\*\*\* //Добавляет узел в конец списка и возвращает указатель // на вставленный узел. template<class mytype> List<mytype>::Node\*

List<mytype>::Add(mytype d){

 Node\* pv=new Node(d); //Создание нового узла

 if(pbeg==0)pbeg=pv; //первый узел списка else {

 Node\* rab=pbeg; while(rab!=0){ if((rab->next)==0){rab->next=pv;return pv;}

 rab=rab->next;

 }

 }

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Node\* Find(mytype key)

//Выполняет поиск узла с заданным ключом и возвращает указатель // на него в случае успешного поиска и 0 в случае отсутствия узла в списке

template<class mytype> List<mytype>::Node\*

List<mytype>::Find(mytype key){

 Node\* pv=pbeg; while(pv){ if((pv->d)==key)break; pv=pv->next;

 }

 return pv;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\* Node\* Insert(mytype key,mytype d)

//Вставляет в список узел после узла с ключом key и возвращает

// указатель на вставленный узел. Если такого узла в списке нет,

// вставка не выполняется и возвращается значение 0

template<class mytype> List<mytype>::Node\* List<mytype>::Insert(mytype key,mytype d){

 if(Node\* pkey=Find(key)) //поиск узла с ключом key

 {

 Node\* pv=new Node(d);

//выделение памяти под новый узел и его инициализация

 pv->next=pkey->next;

//установление связи нового узла с последующим pkey->next=pv; //установление связи предыдущего узла с новым return pv;

 } return 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* bool Remove(mytype key)

//Удаляет узел с заданным ключом из списка и возвращает значение true при //успешном удалении и false, если узел с таким ключом не найден template<class mytype> bool List<mytype>::Remove(mytype key){ if(Node\* pkey=Find(key)){ if(pkey==pbeg)pbeg=pbeg->next; //удаление из начала списка

 else{ //Находим указатель на узел, Node\*rab=pbeg; //стоящий в списке перед while(rab) //удаляемым узлом. { //rab-этот указатель.

 if((rab->next)==pkey)break;

 rab=rab->next;

 }

 rab->next=pkey->next;

 }

 delete pkey; return true;

 } return false;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* void Print() -Печать списка template<class mytype> void List<mytype>::Print(){ Node\*pv=pbeg;

 cout<<Rus("Наш список:");cout<<endl;

 while(pv){ cout<<pv->d<<' ';

 pv=pv->next;

 }

 cout<<endl;}

//--------------------------- MAIN --------------------------------------------- int main(int argc, char\* argv[]){ int k=0,max,kol; char menu[][100]= {{" ListForInt "}, {" ListForFloat "},

 {" ListForDouble "}, {" EXIT "}, };

 kol=4; //КОЛИЧЕСТВО СТРОК МЕНЮ. Это используется в выравнивании

 //строк меню по центру.

//----ВЫРАВНИВАНИЕ СТРОК МЕНЮ ПО ЦЕНТРУ------------------ max=viravnivaniestrok(menu,kol);

//----------------- МЕНЮ НА ЭКРАНЕ--------------------------------------- textmode(C80); while(1){ switch(mmm(kol,menu,max,k))

 { case 0: {

 ListForInt(); k=0;break;

 } case 1: {

 ListForFloat(); k=1;break;

 } case 2: {

 ListForDouble(); k=2;break;

 } case 3:{ exit(0);

 }

 } } return 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* void ListForInt() //Эта функция вызывается из главного меню.

void ListForInt(){ List<int>l1; int k=0,max,kol;

 char menu[][100]=

 { {" PrintList "}, {" Add "}, {" Find "}, {" Insert "},

 {" Remove "}, {" EXIT "}, {" Back "} }; kol=7; //КОЛИЧЕСТВО СТРОК МЕНЮ. max=viravnivaniestrok(menu,kol);

//------------------------ МЕНЮ НА ЭКРАНЕ----------------------------- textmode(C80); while(1){ switch(mmm(kol,menu,max,k))

 { case 0: { l1.Print(); while(!kbhit()) k=0;break;} case 1: {

 cout<<Rus("введите число, которое надо вставить:");

 int t;cin>>t;

 if( (l1.Add(t)) )cout<<Rus("вставка осуществлена");

 else cout<<Rus("вставка не осуществлена"); while(!kbhit()); k=1;break;} case 2: { cout<<Rus("введите искомое число:"); int t; cin>>t;

 if(l1.Find(t))cout<<Rus("искомое число есть в списке.");

 else cout<<Rus("искомого числа нет в списке."); while(!kbhit()); k=2;break;} case 3: {

 cout<<Rus("введите число, которое надо вставить:"); int t;cin>>t;

 cout<<Rus("введите число, после которого надо вставить:"); int key;cin>>key; if( (l1.Insert(key,t)) )cout<<Rus("вставка осуществлена"); else cout<<Rus("вставка не осуществлена"); while(!kbhit()); k=3;break;} case 4: {

 cout<<Rus("введите число, которое надо удалить:"); int t;cin>>t;

 if( (l1.Remove(t)) )cout<<Rus("удаление осуществлено"); else cout<<Rus("такого числа нет в списке."); while(!kbhit()); k=4;break;} case 5:{exit(0);} } } }

Таким же образом, как и функция ListForInt(), реализуются функции ListForFloat() и ListForDouble(), предназначенные для тестирования списков из чисел типа float и double, соответственно.

**Задания для самостоятельного решения**

Для разработки шаблонов классов можно использовать результаты выполнения лабораторных работ № 2 и № 3. При тестировании созданных шаблонов классов необходимо создавать объекты с различными допустимыми значениями параметров шаблона (например, компоненты вектора могут быть целыми, действительными или комплексными числами).

1. Создать шаблон класса для работы со стеком. Применить его для решения задач № 1 – 4 (лаб. работа № 8).
2. Создать шаблон класса для работы с одномерным массивом. Выполнить тестирование путем создания и обработки массивов, содержащих элементы различных типов (например, сортировка элементов массивов различными методами).
3. Создать шаблон класса **Vector** размерности *n* (см. задачу № 3, лаб. работа № 7).
4. Создать шаблон класса «Квадратная матрица» – **Matrix** размерности *n*×*n* (см. задачу № 4, лаб. работа № 7).
5. Создать шаблон класса **Polynom** степени *n* (см. задачу № 5, лаб. работа № 7) или создать шаблон класса для работы с односвязным списком. Применить его для решения задачи № 5 (лаб. работа № 8).
6. Создать шаблон класса для работы с односвязным списком. Применить шаблон класса для решения задачи № 6 (лаб. работа № 8).
7. Создать шаблон класса для работы с односвязным списком. Применить шаблон класса для решения задачи № 7 (лаб. работа № 8).
8. Создать шаблон класса для работы с бинарным деревом. Применить его для сортировки действительных чисел и строк, вводимых с клавиатуры или из файла.
9. Создать шаблон класса **Set** (множество) мощности *n* (см. задачу № 9, лаб. работа № 7 ) или создать шаблон класса для работы с односвязным списком. Применить шаблон класса для решения задачи № 9 (лаб. работа № 8).
10. Создать шаблон класса для работы с односвязным списком. Применить шаблон класса для решения задачи № 10 (лаб. работа № 8).
11. Создать шаблон класса, обеспечивающего описание матрицы заданного размера *n*×*m* и любого минора в ней (см. задачу № 11, лаб. работа № 7) или создать шаблон класса для работы с двусвязным списком. Применить шаблон класса для решения задачи № 11 (лаб. работа № 8).
12. Создать шаблон класса для работы с бинарным деревом. Применить шаблон класса для решения задачи № 12 (лаб. работа № 8).
13. Создать шаблон класса для работы с бинарным деревом. Применить шаблон класса для решения задачи № 13 (лаб. работа № 8).
14. Создать шаблон класса для работы с двусвязным списком. Применить шаблон класса для решения задачи № 15 (лаб. работа № 8).
15. Создать шаблон класса для работы с односвязным списком. Применить шаблон класса для решения задачи № 16 (лаб. работа № 8).

**Лабораторная работа 10. Наследование С++**

**Теоретическое введение.**

Важнейшим принципом ООП, реализованным в С++, является *наследование.* Класс, который наследуется, называется базовым классом, а наследующий класс – производным классом. Указателю на объект базового класса можно присвоить значение указателя на объект производного класса.

Все члены класса из разделов ***public*** и ***protected*** наследуются, а члены из раздела ***private –*** нет***.*** Члены раздела ***protected*** являются частными для базового и производного классов. При наследовании базового класса к его членам также может объявляться спецификатор ***доступ***.

#### class имя\_класса: *доступ* имя\_базового\_класса {/\*члены класса\*/}

Cпецификатор ***доступ*** базового класса при наследовании может принимать одно из трех значений: ***public*** (по умолчанию), ***private, protected.*** Если спецификатор принимает значение ***public,*** то все члены разделов ***public*** и ***protected*** базового класса становятся членами разделов ***public*** и ***protected*** производного класса. Если доступ имеет значение ***private***, то все члены разделов ***public*** и ***protected*** становятся членами раздела ***private*** производного класса. Если доступ имеет значение ***protected***, то все члены разделов ***public, protected*** становятся членами раздела ***protected*** производного класса.

Конструкторы и деструкторы базовых классов не наследуются**,** однако при создании объектов производных классов конструкторы базовых классов выполняются в порядке наследования, а деструкторы в обратном порядке. При необходимости передачи аргументов конструктору базового класса из производного класса используется следующий синтаксис:

####  конструктор\_производного\_класса(арг):base(арг) {/\*тело конструктора производного класса\*/}

**Множественное наследование.** Один класс может наследовать атрибуты двух и более базовых классов, которые перечисляются после двоеточия через запятую. Если базовые классы содержат конструкторы, то они вызываются поочередно в порядке перечисления.

**Виртуальные функции и полиморфизм.** Механизм виртуальных функций в ООП используется для реализации полиморфизма: создания метода, предназначенного для работы с различными объектами за счет механизма позднего связывания *(late binding).* Виртуальные функции объявляются в базовом и производных классах с ключевым словом ***virtual****.* При этом каждый объект класса, управляемого из базового класса с виртуальными функциями, имеет указатель на *vmtbl (virtual method table)*, содержащую адреса виртуальных функций. Эти адреса устанавливаются в адреса нужных для данного объекта функций во время выполнения.

В отличие от перегружаемых функций виртуальные объявляются в порожденных классах с тем же именем, возвращаемым значением и типом аргументов. Если различны типы аргументов, виртуальный механизм игнорируется. Тип возвращаемого значения переопределить нельзя.

Основная идея в использовании виртуальной функции состоит в следующем: она может быть объявлена в базовом классе, а затем переопределена в каждом производном классе. При этом доступ через указатель на объект базового класса осуществляется к этой функции из базового класса, а доступ через указатель на объект производного класса – из производного класса. То же происходит при передаче функции объекта производного класса, если аргумент объявлен как базовый класс.

*Абстрактные классы* – это классы, содержащие *чисто абстрактные виртуальные функции*. Чисто абстрактные виртуальные функции при объявлении в классе приравниваются к нулю. Абстрактные классы используются только для наследования, так как объекты таких классов не могут быть созданы.

**Пример.** Создаются три класса: MyPoint (базовый класс – «Точка»), MyEllipse (производный класс от класса MyPoint – «Окружность») и MyCylinder (производный класс от класса MyEllipse – «Цилиндр»). Классы объявлены в файле MyClass.h. Используются кнопки: Button1 – очистка области для рисования (компонент Image1), Button2 – создание объекта класса MyPoint и рисование точки, Button3 – создание объекта класса MyEllipce и рисование окружности, Button4 – создание объекта класса MyCylinder и рисование цилиндра, компоненты СSpinEdit1 – СSpinEdit4 для задания координат точки и размеров фигур, а также перемещения объектов в области рисования.

**Заголовочный файл** Myclass.h

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* class MyPoint \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* class MyPoint {

 int x,y; public:

 int GetX(){return x;} int GetY(){return y;} void SetX(int x1){x=x1;} void SetY(int y1){y=y1;}

 MyPoint(int xx=0,int yy=0){x=xx;y=yy;}

 virtual void Show(); virtual bool Checking(); // Проверка, не выходит ли точка за пределы области рисования ?

 virtual void MoveTo(int x1,int y1);

};

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*classs MyEllipse \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class MyEllipse:public MyPoint {

 int xa,yb; public:

 int GetXA(){return xa;} int GetYB(){return yb;} void SetXA(int xa1){xa=xa1;} void SetYB(int yb1){yb=yb1;}

 MyEllipse(int a=0,int b=0,int c=0,int d=0):MyPoint(a,b)

 {xa=c;yb=d;} virtual void Show(); virtual bool Checking();

 // Проверка, не выходит ли окружность за пределы области рисования ?

};

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* class MyCylinder \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class MyCylinder:public MyEllipse {

 int h; public:

 int GetH(){return h;} void SetH(int h1){h=h1;}

 MyCylinder(int X=0,int Y=0,int R=0,int H=0):MyEllipse(X,Y,R,R/2)

 {h=H;} void Show(); bool Checking();

// Проверка, не выходит ли цилиндр за пределы области рисования ?

};

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* class MyCylinder \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* bool ExistFigure=false;//Если ExistFigure==false, объект не изображался MyPoint \*Figure;//- Указатель на базовый класс MyPoint. **Файл** Unit1.Cpp

#include <vcl.h>

#pragma hdrstop

#include "Unit1.h"

#include "Myclass.h"

#pragma package(smart\_init)

#pragma link "CSPIN"

#pragma resource "\*.dfm"

TForm1 \*Form1;

\_\_fastcall TForm1::TForm1(TComponent\* Owner): TForm(Owner)

{ }

//\*\*\*\*\*\*\*\*\*\* Реализация класса MyPoint \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* void MyPoint::Show()

 ( Form1->Image1->Canvas->Pixels[x][y]=clRed;} bool MyPoint::Checking(){

//- Проверка, не выходит ли точка за пределы области рисования ?

if((x>=0)&&(x<=Form1->Image1->Width)&&(y>=0)&&

(y<=Form1->Image1->Height))

 return true; else return false;

 }

 void MyPoint::MoveTo(int x1,int y1)

 { // если выходим за область рисования, то точку не сдвигаем.

 int a=x,b=y; x=x1;y=y1;

if(!Checking()){x=a;y=b;}

 }

//\*\*\*\*\*\*\*\*\*\*\*Реализация класса MyEllipse \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* void MyEllipse::Show()

 {

Form1->Image1->Canvas->Ellipse(GetX()-xa,GetY()-yb,GetX()+xa,GetY()+yb);

 }

// Проверка, не выходит ли окружность за пределы области рисования ? bool MyEllipse::Checking()

{ int rabx=GetX(),raby=GetY(); SetX(GetX()-xa);SetY(GetY()-yb); if(MyPoint::Checking())

 {

 SetX(GetX()+2\*xa);SetY(GetY()+2\*yb);

 if(MyPoint::Checking())

 {

 SetX(rabx);SetY(raby); // восcтанавливаем координаты x,y. return true;

 } }

 SetX(rabx);SetY(raby); // восcтанавливаем координаты x,y. return false; }

//\*\*\*\*\*\*\*\*\*\*Реализация класса MyCylinder \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* void MyCylinder::Show()

 {

 MyEllipse::Show();

 Form1->Image1->Canvas->MoveTo(GetX()-GetXA(),GetY());

 Form1->Image1->Canvas->LineTo(GetX()-GetXA(),GetY()-h);

 Form1->Image1->Canvas->MoveTo(GetX()+GetXA(),GetY());

 Form1->Image1->Canvas->LineTo(GetX()+GetXA(),GetY()-h);

 SetY(GetY()-h);

 MyEllipse::Show();

 SetY(GetY()+h);

 }

bool MyCylinder::Checking()

{

 if(MyEllipse::Checking())

 {

 SetY(GetY()-h);

 if(MyEllipse::Checking()){SetY(GetY()+h);return true;}

 } return false;

 }

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{ // Чистим "графическое" окно

 Image1->Canvas->FillRect(Rect(0,0,Image1->Width,Image1->Height));

 ExistFigure=false;

}

void \_\_fastcall TForm1::Button2Click(TObject \*Sender)

{

 if(ExistFigure==true)delete Figure;

 Image1->Canvas->FillRect(Rect(0,0,Image1->Width,Image1->Height));

 // Рисуем точку, проверив её попадание в область рисования

 Figure=new MyPoint(CSpinEdit1->Value,CSpinEdit2->Value);

 ExistFigure=true;

 if(Figure->Checking()) Figure->Show(); else ShowMessage("Точка не попадает в область рисования!");

}

void \_\_fastcall TForm1::Button3Click(TObject \*Sender)

{

 if(ExistFigure==true) delete Figure;

 Image1->Canvas->FillRect(Rect(0,0,Image1->Width,Image1->Height));

 // Рисуем окружность, проверив её попадание в область рисования

 Figure=new MyEllipse(CSpinEdit1->Value,CSpinEdit2->Value,

CSpinEdit3->Value,CSpinEdit3->Value); ExistFigure=true;

 if(Figure->Checking())Figure->Show(); else ShowMessage("Окружность не помещается в область рисования!");

}

void \_\_fastcall TForm1::Button4Click(TObject \*Sender)

{

 if(ExistFigure==true) delete Figure;

 Image1->Canvas->FillRect(Rect(0,0,Image1->Width,Image1->Height));

 // Рисуем цилиндр, проверив его попадание в область рисования

 Figure=new MyCylinder(CSpinEdit1->Value,CSpinEdit2->Value,

CSpinEdit3->Value,CSpinEdit4->Value); ExistFigure=true;

 if(Figure->Checking())Figure->Show(); else ShowMessage("Цилиндр не помещается в область рисования!");

}

void \_\_fastcall TForm1::CSpinEdit1Change(TObject \*Sender)

{

 if(ExistFigure==true){ Image1->Canvas->FillRect(Rect(0,0,Image1->Width,Image1->Height)); Figure->MoveTo(CSpinEdit1->Value,CSpinEdit2->Value);

 Figure->Show();

 } }

void \_\_fastcall TForm1::CSpinEdit2Change(TObject \*Sender)

{

 if(ExistFigure==true){

 Image1->Canvas->FillRect(Rect(0,0,Image1->Width,Image1->Height));

 Figure->MoveTo(CSpinEdit1->Value,CSpinEdit2->Value);

 Figure->Show();

 }

}

**Задания для самостоятельного решения**

При выполнении данной работы необходимо определить базовый класс и производные от него классы. Предусмотреть передачу аргументов конструкторам базового класса; использование виртуальных и перегруженных функций; обработку исключительных ситуаций. *Вариант А*

В следующих заданиях требуется создать базовый класс (как вариант абстрактный базовый класс) и определить общие методы show ( ), get ( ), set ( ) и другие, специфические для данного класса. Создать производные классы, в которые добавить свойства и методы.

Часть методов переопределить. Создать массив объектов базового класса и заполнить объектами производных классов. Объекты производных классов идентифицировать конструктором по имени или идентификационному номеру.

Вызвать метод show ( ) базового класса и просмотреть массив объектов.

Использовать объекты для моделирования реальных ситуаций.

1. Создать базовый класс «Транспортное средство» и производные классы «Автомобиль», «Велосипед», «Повозка». Подсчитать время и стоимость перевозки пассажиров и грузов каждым транспортным средством.
2. Создать базовый класс «Грузоперевозчик» и производные классы «Самолет», «Поезд», «Автомобиль». Определить время и стоимость перевозки для указанных городов и расстояний.
3. Создать аналогичный базовый класс «Пассажироперевозчик» и производные классы «Самолет», «Поезд», «Автомобиль». Определить время и стоимость передвижения.
4. Изменить задания 1–3, чтобы базовый класс стал абстрактным. Сделать некоторые методы абстрактными.
5. Создать базовый класс «Учащийся» и производные классы «Школьник» и «Студент». Создать массив объектов базового класса и заполнить этот массив объектами. Показать отдельно студентов и школьников.
6. Создать базовый класс «Музыкальный инструмент» и производные классы «Ударный», «Струнный», «Духовой». Создать массив объектов «Оркестр». Выдать состав оркестра, переопределив метод.
7. Определить базовый класс «Множество» и производный класс «Кольцо» (операции сложения и умножения обе коммутативные и ассоциативные, связанные законом дистрибутивности; сложение обладает обратной операцией – вычитанием). Ввести кольца целых чисел, многочленов, систему классов целых чисел, сравнимых по модулю. Кольцо является полем, если в нем определена операция деления, кроме деления на нуль. Рациональные числа, дробно рациональные функции.
8. Создать абстрактный класс «Работник фирмы» и производные классы «Менеджер», «Администратор», «Программист».
9. Создать базовый класс «Домашнее животное» и производные классы «Собака», «Кошка», «Попугай» и др. С помощью конструктора установить имя каждого животного и его характеристики.
10. Создать базовый класс «Садовое дерево» и производные классы «Яблоня», «Вишня», «Груша» и др. С помощью конструктора автоматически установить номер каждого дерева. Принять решение о пересадке каждого дерева в зависимости от возраста и плодоношения. *Вариант Б*
11. Создать класс Item (единица хранения в библиотеке), содержащий данные-члены: invNumber – инвентарный номер и taken – взято на руки или имеется в наличии, а также методы:

virtual void Show(); //показать информацию о единице хранения bool isAvailable(); // есть ли единица хранения в наличии ?

int GetinvNumber(); //возвращает инвентарный номер

void Take(); // операция «взять»

void Return(); // операция «вернуть»

Построить производные классы Book и Magazin. Класс Book содержит данные-члены: author, title, publisher, year и методы: Author(); Title(); Publisher(); YearOf Publishing(); Show().

Класс Magazin включает данные-члены: volume; number; year; title и методы: Volume(); Title(); Number(); Year(); Show().

1. Создать базовый класс Polygon (многоугольник). Класс должен содержать методы для рисования многоугольника, вычисления периметра, нахождения площади и др. Построить производный класс Triangle (треугольник), содержащий также методы для нахождения точки пересечения медиан, длин медиан, длин биссектрис, координат точек пересечения биссектрис, высот треугольника.
2. Создать абстрактный класс Shape для рисования плоских фигур. Построить производные классы Square (квадрат, который характеризуется координатами левого верхнего угла и длиной стороны), Circle (окружность с заданными координатами центра и радиусом), Ellipse (эллипс с заданными координатами вершин описанного вокруг него прямоугольника), позволяющие рисовать указанные фигуры, а также передвигать их на плоскости.
3. Создать класс CPoint – точка и производные от него классы CcoloredPoint и CLine. На основе классов CcoloredPoint и CLine создать класс CcoloredLine. Все классы должны иметь методы для установки и получения значений всех координат, а также изменения цвета и получения текущего цвета.
4. Описать базовый класс Stroka. Обязательные данные-члены класса: указатель типа char – для хранения строки; значение типа int – длина строки.

Методы: конструктор без параметров; конструктор, принимающий в качестве параметра C-строку (заканчивается нулевым байтом); конструктор копирования; получение длины строки; очистка строки (сделать строку пустой); деструктор.

Описать производный класс «БИТОВАЯ\_СТРОКА» (строки данного класса могут содержать только символы ‘0’ и ‘1’). Если в основе инициализирующей строки встретятся любые символы, отличные от допустимых, то БИТОВАЯ\_СТРОКА становится пустой. Содержимое строки рассматривается как двоичное представление целого числа со знаковым разрядом. Отрицательные числа хранятся в дополнительном коде.

Обязательные методы: конструктор без параметров; конструктор, принимающий в качестве параметра C-строку; конструктор копирования; деструктор; изменение знака числа (перевод числа в дополнительный код).

Переопределить следующие операции (длина строки результата в случае необходимости расширяется влево знаковым разрядом): присваивание; сложение (+); проверка на равенство (==).

1. Создать производный класс «СТРОКА10» (целое неотрицательное десятичное число) от класса «СТРОКА» (описание приведено выше).

Методы: конструктор без параметров; конструктор, принимающий в качестве параметра C-строку; конструктор копирования; деструктор; метод, определяющий, можно ли представить данное число в формате int; метод, определяющий, равно ли число нулю; метод, возвращающий представление числа в виде целого (int); метод, удаляющий незначащие нули.

Переопределить операции: сложение (+); проверка на больше (по значению) (>); проверка на меньше (<); присваивание (=).

1. Создать производный класс «БУЛЕВ ВЕКТОР» (**BoolVector**) от класса **Vector**. Компоненты принимают значения из множества {0,1}.

Методы: конструктор без параметров; конструктор, принимающий в качестве параметров указатель на массив целого типа (если элементы массива содержат числа, отличные от 0 и 1, то создается пустой вектор) и размер вектора; конструктор копирования; деструктор; метод, возвращающий число единиц в векторе; метод, возвращающий позицию самой левой единицы в векторе.

Переопределить операции: поразрядная конъюнкция (&); поразрядная дизъюнкция (|); поразрядная инверсия (~); поразрядная операция ИСКЛЮЧАЮЩЕЕ ИЛИ (^); присваивание (=).

1. Создать производный класс «ТРОИЧНЫЙ ВЕКТОР» от класса **Vector**. Компоненты принимают значения из множества {0,1,2}.

Методы: конструктор без параметров; конструктор, принимающий в качестве параметров указатель на массив целого типа и размер вектора; конструктор копирования; деструктор; проверка двух векторов на ортогональность (два троичных вектора называются ортогональными, если в них существует пара одноименных компонент, имеющих в одном из векторов значение 0, а в другом – 1); метод, возвращающий число компонент в векторе, принимающих значение 2.

Переопределить операции: присваивание (=); поразрядная конъюнкция (пересечение) двух не ортогональных векторов (&): 0&0=0, 1&1=1, 2&2=2, 0&2=0, 2&0=0, 1&2=1, 2&1=1; индексирование ([]).

1. Создать производный класс «БУЛЕВА МАТРИЦА» от класса «ЦЕЛОЧИСЛЕННАЯ МАТРИЦА».

Методы: конструктор без параметров; конструктор, принимающий в качестве параметров целочисленный двумерный массив, содержащий матрицу, и ее размеры *n* и *m*; конструктор копирования; деструктор; метод возвращает число единиц в матрице; метод, возвращающий α-каноническую матрицу (в исходной матрице удалены повторяющиеся строки; строки составлены в порядке возрастания неотрицательных чисел, в качестве двоичных кодов которых рассматриваются данные строки).

Переопределить операции: поэлементная конъюнкция двух матриц (&); поэлементная дизъюнкция двух матриц (|); поэлементная операция ИСКЛЮЧАЮЩЕЕ ИЛИ двух матриц (^); произведение двух мат-

риц A=[aij] и B=[ bjk], где i=1,*n*, j=1,*m* , k=1,*l* (\*). При вычислениях операция целочисленного умножения заменяется конъюнкцией, а сложение – дизъюнкцией; присваивание (=).

1. Создать производный класс «МИНОР» для базового класса «МАТРИЦА» размерности *n*×*m*. Переопределить для производного класса операции и методы (см. лаб. работу № 2, задание № 4).
2. Расширить возможности стандартного класса Time, чтобы можно было выводить время дня: утро, вечер и т. д.
3. Расширить возможности стандартного класса Date, чтобы можно было выводить время года: зима, лето и т. д.
4. Расширить возможности класса Annotation, чтобы можно было выводить время и дату изменения аннотации.
5. Расширить возможности класса Dictionary, чтобы можно было выводить дату последнего изменения в словаре.
6. Расширить возможности класса File, чтобы можно было выводить время и дату создания файла.
7. Расширить возможности класса Stack, чтобы можно было выводить время последнего сеанса работы со стеком.
8. Определить базовый класс для работы с прямоугольными матрицами, предусмотрев ввод-вывод матриц и выполнение следующих операций: сложение матриц; умножение матрицы на скаляр; перестановка строк матрицы по заданному вектору транспозиции; перестановка столбцов матрицы по заданному вектору транспозиции. В производном классе реализовать указанные операции для квадратных матриц, добавив выполнение следующих операций:транспонирование матрицы; умножение матриц.

**Лабораторная работа 11. Потоки, обработка исключительных ситуаций в C++**

**Теоретическое введение.**

В С++ ввод-вывод осуществляется через потоки. Потоки являются объектами соответствующих классов. При запуске программы автоматически открываются стандартные потоки ***cin, cout, cerr, clog.*** Последние два потока используются для вывода сообщений об ошибках. В файле ***iostream.h*** определены классы: ввода – ***istream***, вывода – ***ostream***, ввода-вывода – ***iostream****.*

Для реализации файлового ввода-вывода небходимо включить файл ***fstream.h***, содержащий производные от ***istream*** и ***ostream*** классы ***ifstream,******ofstream*** и ***fstream***,и объявить соответствующие объекты.

Например:

**ifstream in;//ввод ofstream out;//вывод fstream io;//ввод-вывод**

После объявления потоков производится открытие файла, связывающее его с потоком с помощью функции ***open()*** или с помощью конструктора. Прототип функции ***open():***

**void open (char \*filename,int mode,int access);**

Здесь **filename** – имя файла, включающее путь; ***mode*** – режим открытия файла (**ios::in** – открытие файла для чтения, **ios::out** – открытие для записи, **ios::binary** – открытие файла в двоичном режиме, по умолчанию в текстовом); **access:** 0 – файл со свободным доступом, 1 – только для чтения, 8 – архивный файл. Файл закрывается с помощью функции ***close().***

Для чтения-записи здесь можно использовать перегружаемые оператор-функции >> и << или использовать методы классов. Для ввода-вывода одного символа используются функции:

**istream &get(char &ch); ostream &put(char ch);**

Для записи и считывания блоков двоичных данных используются функции считывания-записи *n* байт в буфер или из буфера:

**istream &read(unsigned char \*buf, int n); ostream &write(const unsigned char \*buf, int n);**

*Обработка исключительных ситуаций.*В программах на С++ следует использовать механизм обработки исключительных ситуаций. Операторы программы при обработке исключительных ситуаций располагаются в блоке ***try***. Перехватывается и обрабатывается исключительная ситуация в блоке ***catch***. Форма операторов ***try-catch*** следующая:

**try {/\*блок try\*/ } catch(type1 arg){/\*блок catch\*/}**

С блоком ***try*** можетсвязываться несколько блоков ***catch***. Выполняется тот блок ***catch***, для которого тип аргумента соответствует типу возникшей исключительной ситуации. При этом ее значение присваивается аргументу ***catch***. Если ошибка имеет место внутри блока ***try***, она может генерироваться с помощью ***throw***, после чего управление передано блоку ***catch.*** В случае необходимости перехвата всех исключительных ситуаций независимо от типа используется **catch(…)**

**{/\*тело\*/}**

Для функций, вызываемых из блока ***try***, можно указать число типов исключительных ситуаций, которые будет генерировать функция:

**тип имя (список аргументов) throw(список типов) {/\*тело\*/}**

**Пример.** Из текстового потока ввести информацию о студентах и записать ее в виде дерева. Отсортировать или по фамилии, или по баллу (вывести тех студентов, у которых балл выше среднего).

Использовать потоки ввода-вывода и шаблоны.

#include <iostream.h>

#include <conio.h> #include <fstream.h> struct Student{ int num; char surname[10]; int group; int balls;

 friend ostream &operator<< (ostream &stream, Student stud){ stream << " " << stud.num << " " << stud.surname << " " << stud.group

 << " " << stud.balls; return stream;

 }

 friend istream &operator>> (istream &stream, Student &stud){

 stream >> stud.num >> stud.surname >> stud.group >> stud.balls; return stream;

 } }; struct node{

 Student info; node \*nextl, \*nextr;

 node (){ info.num = info.group = info.balls=0;

 nextl = nextr = 0;

 }

 node (Student newinfo){ info = newinfo; nextl = nextr = 0;

 } }; template <class T, class T1> class tree{ public: T \*root; tree() { root = 0; } void push (T \*& wer, T1 dat, int n){

 if (wer == 0){ try{ wer = new T; if(!wer) throw 1;

 wer->nextl = 0; wer->nextr = 0; wer->info = dat;

 }

 catch (int mthrow) {cout<<”No memory!”<<endl;return;}

 } else if (n == 1) if (strcmp(dat.surname,wer->info.surname) < 0) push (wer->nextl, dat, 1); else push (wer->nextr, dat, 1); else if (dat.balls > wer->info.balls) push (wer->nextl, dat, 2); else push (wer->nextr, dat, 2);

 }

 void insert (T1 dat, int n){ if (root == 0) root = new T(dat); else push (root, dat, n);

 }

 void look (ostream &stream, T \*&wer){

 if (wer != 0){

 look (stream, wer->nextl); stream << " " << wer->info << endl; look (stream, wer->nextr);

 }

 }

 friend ostream &operator<< (ostream &stream, tree ob)

 { ob.look (stream, ob.root); return stream; }

}; void main(){ int m; do{ cout << "1. Sort with names\n"; cout << "2. Sort with balls\n"; cout << "3. Exit\n"; cin >> m; switch (m){

 case 1: { tree<node, Student> q;

 node \*n; ifstream infile("stud.txt");

 while(!infile.eof()){ Student c; infile >> c;

 q.insert(c, 1); } infile.close(); cout<<q; break; } case 2: { tree<node, Student> q; node \*n;

 ifstream infile("stud.txt"); Student \*c; c = new Student;

 int i = 1; float s = 0; while(!infile.eof()){ infile >> c[i]; s+=c[i].balls;

 i++;

 }

 for (int j=1; j<=i; j++) if (c[j].balls > s/i)

 q.insert(c[j], 2); infile.close(); cprintf(" Miide ball is %1.3f",s/i); cout<<'\n' << q;

 break;

 }

 case 3: {return;} default: {cout<<"Error! Try again\n"; break;}

 } getch(); clrscr(); }

 while(m != 3); return;}

**Задания для самостоятельного решения**

При выполнении приводимых ниже заданий можно использовать классы, разработанные в лабораторных работах № 1–3. Осуществлять контроль состояния потоков. В случае возникновения ошибок потоков генерировать и обрабатывать исключительные ситуации. Для соответствующих классов перегрузить операции вставки в поток и извлечения из потока. При динамическом выделении памяти предусмотреть обработку исключения, возникающего при нехватке памяти.

I

а) Для класса **Student** (лаб. работа № 1) предусмотреть ввод

данных из файла. Полученные при выполнении лаб. работы № 1 списки студентов вывести в файл. То же задание для классов:

б) **Abiturient** (лаб. работа №1);

в) **Aeroflot** (лаб. работа № 1);

г) **Worker** (лаб. работа № 1);

д) **Train** (лаб. работа № 1);

е) **Product** (лаб. работа № 1);

ж) **Patient** (лаб. работа № 1);

з) **Bus** (лаб. работа № 1);

и) **Customer** (лаб. работа № 1);

к) **File** (лаб. работа № 1);

л) **Word** (лаб. работа № 1);

м) **House** (лаб. работа № 1);

н) **Phone** (лаб. работа № 1);

о) **Person** (лаб. работа № 1).

II

а) При выполнении задания № 1 лаб. работы № 2 (класс **Complex**) предусмотреть формирование массива объектов путем считывания комплексных чисел из файла. Результат также вывести в файл. То же задание для классов:

б) **Fraction** (лаб. работа № 7);

в) **Vector** (лаб. работа № 7). Предусмотреть обработку исключе-

ния при динамическом выделении памяти;

г) **Matrix** (лаб. работа № 7);

д) **Polynom** (лаб. работа № 7);

е) **Stack** (лаб. работа № 7);

ж) **Строка** (лаб. работа № 7);

з) **Set** (лаб. работа № 7);

и) **«Массив строк»** (зад. № 10 лаб. работы № 2);

к) «**Булев вектор»** (лаб. работа № 7);

л) «**Троичный вектор»** (лаб. работа № 7);

м) «**Булева матрица»** (лаб. работа № 7).

III

Те же задания, что и в разделах I и II, но для классов, реализующих работу с динамическими структурами данных (см. лаб. работу № 3).

**Лабораторная работа 12. Работа с типом данных структура.**

**Цель занятия:**

* Изучение алгоритмов обработки объектов структурного типа данных и способов их реализации в языке С++.

**Заданя для лабараторной работы**

Описать структуру с именем STUDENT, содержащую следующие поля:

* Фамилия и инициалы;
* Номер группы;
* Успеваемость (массив из 5 элементов).

Написать программу, выполняющую следующие действия:

* Ввод с клавиатуры данных в массив, состоящий из 10 структур типа STUDENT; записи должны быть упорядочены по возрастанию номера группы;
* Вывод фамилий и номеров групп тех студентов, чей средний балл больше 4.0.
* Если таких студентов нет, вывести соответствующее сообщение.

Описать структуру с именем AEROFLOT, содержащую следующие поля:

* Название пункта назначения рейса;
* Номер рейса;
* Тип самолёта.

Написать программу, выполняющую следующие действия:

* Ввод с клавиатуры данных в массив, состоящий из 7 структур типа AEROFLOT; записи должны быть упорядочены по возрастанию номера рейса;
* Вывод номеров рейсов и типов самолётов, вылетающих в пункт назначения, название которого совпало с названием, введённым с клавиатуры.
* Если таких рейсов нет, вывести соответствующие сообщение.

Описать структуру с именем NOTE содержащую следующие поля:

* Фамилия, имя;
* Номер телефона;
* Дата рождения (массив из трёх чисел).

Написать программу, выполняющую следующие действия:

* Ввод с клавиатуры данных в массив, состоящий из 8 структур типа NOTE, записи должны быть упорядочены по датам рождения;
* Вывод информации о тех людях, чьи дни рождения приходятся на месяц, значение которого введено с клавиатуры.
* Если таких нет, вывести соответствующие сообщение.

Описать структуру с именем ZNAK содержащую следующие поля:

* Фамилия, имя;
* Знак Зодиака;
* Дата рождения (массив из трёх чисел).

Написать программу, выполняющую следующие действия:

* Ввод с клавиатуры данных в массив, состоящий из 8 структур типа ZNAK, записи должны быть упорядочены по датам рождения;
* Вывод информации о человеке, чья фамилия введена с клавиатуры;
* Если такого нет, вывести соответствующие сообщение.

**Лабораторная работа 13. Коллекции. Параметризованные классы.**

**Цель:** научиться разрабатывать коллекции на языке программирования C# и применять их в программах.

**Пример 1.** Пример применения параметризованной коллекции List <T>.

В предложенном примере реализована коллекция на основе параметризованной коллекции List<Т>. Коллекция mas содержит список (аналог одномерного массива) экземпляров (объектов) классов типа Monster. В данной коллекции mas можно хранить элементы класса Monster, а также любого класса, производного от Monster. Другие типы данных в коллекции mas недопустимы.

Проект **Lab16\_Фамилия**



**Задание 1.** Создайте в проекте **Lab16\_Фамилия** коллекцию mas1, содержащую список объектов типа int. Добавьте в коллекцию три элемента, ваши данные: Число\_рождения, Месяц\_рождения, Год\_рождения. Выведите элементы коллекции на экран.

**Задание 2.** Создайте в проекте **Lab16\_Фамилия** коллекцию mas2, ваши данные: Фамилия, Имя, Отчество. Выведите элементы коллекции на экран.

**Задание 3.** Создайте в проекте **Lab16\_3\_Фамилия** параметризованный метод Sort с параметром – типом данных. Реализуйте в нем **сортировку методом выбора** (алгоритм ее состоит в том, что сначала выбирается наименьший элемент массива и меняется местами с первым элементом. Затем просматриваются элементы, начиная со второго, и наименьший из них меняется местами со вторым элементом и так далее. Всего произведено будет n-1 замен. На последнем проходе цикла при необходимости меняются местами предпоследний и последний элементы массива). На данный метод накладывается ограничение – объекты можно сравнивать друг с другом с помощью метода CompareTo.

Отсортируйте с помощью данного метода следующие масвы:

a = { 1, 6, 4, 2, 7, 5, 3 }
b = { 1.1, 6.6, 4.4, 2.4, 7.6, 5.5, 3.3 }
c = { sdef, sd, sdfsd, sdf }

**Лабораторная работа 14. Использование регулярных выражений.**

**Цель работы.**

Изучить формирование программ с использованием регулярных выражений. Научиться создавать формы с использованием элемента таймер.

**Теоретические сведения.**

Регулярные выражения.

Назначение регулярных выражений состоит в возможности выделения из строки символов подстрок, соответствующих шаблону. Например для шаблона \bm\s\* будут выделены все подстроки, перед которыми имеется пробел (метасимвол \b), подстроки будут начинаться с буквы m и иметь любой набор символов (метасимвол \s) произвольной длины (знак\*). Шаблон формируется с использованием метасимволов.

Рассмотрим сначала как строится шаблон

1. Шаблон – это строка string, состоящая из набора специальных символов, конструкций и произвольного текста. Произвольный текст это любая последовательность символов из кодовой таблицы.
2. Та как часть символов кодовой таблицы зарезервирована под специальные символы шаблона (метосимволы) но в произвольном тексте эти символы должны указываться c символом \. Пример: \( – т. к. скобка является метосимволом, то в произвольном тексте она должна записываться именно так. Вот перечень этих символов: \ [ ] ( ) { } . \* + ? | ^ $.

Кроме этих символов вместе с \ указываются специальные символы, не отображаемые в редакторе и являющиеся служебными: \n – символ новой строки (этот символ имеется в строке, занимает байт, имеет код 13, но естественно никак не отображается )

\r – символ возврата коретки (код 10)

\t – символ табуляции

\v – символ вертикальной табуляции

\f – символ перевода страницы

Синтаксис шаблона состоит из фрагментов и комментариев. Комментарии не являются частью шаблона и обозначаются конструкцией (?# текст комментария). Фрагмент может в шаблоне выделяться в виде неименованного фрагмента:

Фрагмент1 ( фрагмент2) фрагмент3

где фрагмент 2 выделен в шаблоне, или в виде именованного фрагмента:

фрагмент 1 (?<pole1> фрагмент 2) фрагмент 3

где фрагмент 2 поименован строкой pole1.

Именованные фрагменты необходимы в двух случаях:

1. Если в выделенной строке по шаблону надо потом делать ещё выделение фрагмента 2
2. Если фрагмент 2 несколько раз используется в шаблоне. В этом случае это делается так:

ф1 (?<pole 1>ф2) ф3 (\k<pole1>) ф4

На место «\k» в шаблоне будет вставлен фрагмент ф2.

Фрагмент шаблона можно повторно указывать через \2, где 2 – номер по порядку фрагмента, начиная с 1

НАПРИМЕР : ф1(ф2)ф3(?<a>ф4)ф5(ф6)ф7\2 ф8 => на место комбинации \2 будет подставлен второй фрагмент ф4.

Фрагмент шаблона может включать следующие метасимволы, указания типа символа:

\cx – соответствует в кодовой таблице управляющему символу Сtrl +x:

Пример: \cM – управляющий символ Ctrl+ M

\d – соответствует любой цифре;

\D – соответствует любому символу кроме цифры ;

\w – соответствует букве, цифре или знаку подчеркивания;

\W – соответствует любому символу кроме буквы, цифры и знака подчеркивания;

\s – соответствует символу пробел или символу форматирования (\t\ n и т.п.);

\ S – соответствует любому символу кроме специального;

\ xnm – шестнадцатеричная Escape последовательность ASCII-кода с номером nm:

\xАВ

Значения n и m надо указывать обязательно, даже если n = 0.

**.** – любой символ, кроме конца строки \n

[послед.] – любой одиночный символ из последовательности, записанной внутри []:

[abc] – любой из символов abc. Если символы идут подряд, то последовательность указывается диапазоном:

[ac – z2-7] 2-7 – диапазон цифр от 2 до 7; c-z – диапазон букв от c до z;

Порядок символов соответствует кодовой таблице символов

[^послед.] – любой из символов, кроме тех, которые указаны в последовательности

Следующие метасимволы являются повторителями символа или фрагмента. Эти метасимволы записываются за символом или фрагментом.

символ{n} – соответствует повторению символа или фрагмента (фр) ровно n раз:

(abc){3} равносильно записи abc abc abc

Симв{n,} - соответствует повторению символа как минимум n раз.

Симв{,m} – соответствует повторению символа не больше m раз.

Симв{n,m} – соответствует повторению символа от n до m раз.

\* – ноль и более повторений предыдущего символа.

+ – одно и белее повторений предыдущего символа.

? – ни одного или одно повторение предыдущего символа.

МЕТАСИМВОЛЫ ПОЗИЦИОНИРОВАНИЯ.

\ bсимвол – соответствуют только символу перед которым стоит пробел. Не путать эту конструкцию с \_символ, т. к. в последнем случае будет отбираться строка вместе с пробелом.

Символ\b – выделение символа, за которым идет пробел. Если при записи \b возникает двухсмысленность:

аbс\bdef

(непонятно \b для с или d), то надо использовать круглые скобки: аb(с\b)def – здесь очевидно \ b используется для с буквы.

\В – соответствует середине слова:

аbс(def)\Вqh

соответствует def только в середине слова или строки

^ символ – символ только в начале строк, т. е. после \n.

$символ  – символ только в конце строки, т. е. перед \n.

\Асимвол – символ следует искать только перед первой строкой.

\Zсимвол – символ следует искать только в конце последней строки.

МЕТОСИМВОЛЫ ВЫБОРА ВАРИАНТОВ:

(?: фр1|фр2|фр3) – соответствие фр1 или фр2 или фр3.

символ(?=фр1|фр2|фр3) – соответствует символ, если за ним через пробел идут фр1 или фр2 или фр3:

(животное)(?кот|собака)

Соответствует фрагменту животное, если за этим словом через пробел будут слова кот или собака

символ(?!фр1(фр2|фр3) – соответствует символу, если за ним через пробел идут любое слово кроме фр1 или фр2 или фр3.

РАБОТА С ШАБЛОНАМИ

1. Занести шаблон в строку

string Shablon =@“(\w+)\_=\_\$(\d=+)[.;]”; // это шаблон вида “вата\_=\_5$”.

1. Объявить объект класса Regex с инициализацией шаблона Shablon:

Regex r = new Regex(Shablon);

1. Задать строку text, из которой будут извлекаться подстроки по шаблону Shablon:

string text = “вата\_=\_$5; ручка\_=\_$3;”;

1. Создать объект класса Match, в который занести все множество подстрок в строке text, совпадающих с шаблоном в объекте r:

Match ms = r.Match(text);

1. Проверить, есть ли в ms подстроки:

if(ms.Success){….}

1. Извлекается подстрока из ms

string s= ms.Groups[0].ToString();

ms может содержать несколько подстрок, удовлетворяющих шаблону, но в начале указатель в ms расположен на 1-й подстроке. Индекс [0] указывает на первую подстроку.

7. Обрабатываем подстроку S (например печатаем её и т. п.).

8. Перемещаем указатель в ms на следующую подстроку:

ms=ms.NextMatch();

9. переход на п.5

Извлечение из подстроки ms по фрагментам шаблона, заключенных в шаблоне Shablon в скобки (фр1):

String s1= ms.Groups[2].Tostring(); – будет извлекаться фрагмент шаблона, скобки (фр1) которого в шаблоне shablon вторые по счету, включая именованные фрагменты.

Для именованных фрагментов шаблона извлечения делается в виде:

s1= ms.Groups(''fr1'').ToString();

где '' fr1'' – имя фрагмента, указанного в шаблоне (?<fr1>фр1)

Шаблон разбиения строки на подстроки по заданному в шаблоне набору разделителей:

1. Формируем шаблон разделителей:

string shablon = “[.,:;\_]+”;

1. Объявляем объект класса Regex:

Regex r=new Regex(shablon);

1. Объявляем строку, откуда будут извлекаться подстроки:

String text = “a=1; b=5; c=4”;

1. Объявляем массив строк куда будут заноситься подстроки из строки text по шаблону shablon:

string [] slist = new string[50];

1. Заносим в массив строк slist подстроки из text:

slist = r.Split(text);

1. Работаем с slist как с обычным массивом строк.

Замена в исходной строке text одного фрагмента на другой:

string text1 = Regex.Replace(text,”Маша”,”Нина”);

Строка text1 будет копией text с заменой в text всех слов «Маша» на «Нина»

Проверка наличия в строке text подстроки с заданным шаблоном:

1. Формируем по шаблону shablon объект r класса Regex, как это было показано выше.
2. Задаем проверяемую строку text.
3. Проверяем наличие в text шаблона r:

if (r.IsMatch(text)) { // есть совпадения с шаблоном}.

**Прядок выполнения работы.**

Создать формы:

FORM2 FORM1

Воспроизвести код программы.

// FORM1

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

public Form2 f2 = new Form2();

private void button2\_Click(object sender, EventArgs e)

{

this.Close();

}

private void button1\_Click(object sender, EventArgs e)

{

f2.ShowDialog();

double x, y,z; string st;

z = Convert.ToDouble(f2.textBox1.Text);

x = Convert.ToDouble(textBox1.Text);

y = Math.Sin(x)+66+z;

st = String.Format("{0,4:0.##}",y);

textBox2.Text = st;

}

} }

// FORM2

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.Text.RegularExpressions;

namespace WindowsFormsApplication1

{

public partial class Form2 : Form

{

bool kod;

Timer tm = new Timer();

public Form2()

{

InitializeComponent();

tm.Interval = 300;

this.tm.Tick += new System.EventHandler(this.tm\_Tick);

}

private void button1\_Click(object sender, EventArgs e)

{

// останавливается таймер

// если правильные данные, то окно закрывается

this.label1.ForeColor = Color.Black;

tm.Stop();

this.label1.Visible = true;

if(kod) this.Close();

}

// Таймер через интервал времени вызывает это событие

// В нем в режиме флип-флоп изменяетсЯ видимость метки, чтобы она мигала

private void tm\_Tick(object sender, EventArgs e)

{

this.label1.Visible = !this.label1.Visible;

tm.Start();

}

private void textBox1\_MouseLeave(object sender, EventArgs e)

{ kod=true;

Regex r= new Regex(@"[^\d,+-]|([,\d][+-])|[+-]{2.}|(\d\*,\d\*,\d\*)",RegexOptions.IgnoreCase);

if (r.IsMatch(textBox1.Text))

{

// неправильный формат данных в окгне textBox1

// стартуется таймер для мигания надписи X=

kod = false;

this.label1.ForeColor = Color.Red;

tm.Start();

}

}

} }

**Список рекомендуемой литературы**

**Список основной литературы**

Объектно-ориентированное программирование на Visual Basic в среде Visual Studio .NET : учеб. пособие / В.Н. Шакин, А.В. Загвоздкина, Г.К. Сосновиков. — М. : ФОРУМ : ИНФРА-М, 2018. — 398 с. — (Среднее профессиональное образование). - Режим доступа: http://znanium.com/catalog/product/961516

**Список дополнительной литературы**

Программирование на языке высокого уровня. Программирование на языке С++: учеб. пособие / Т.И. Немцова, С.Ю. Голова, А.И. Терентьев ; под ред. Л.Г. Гагариной. — М. : ИД «ФОРУМ» : ИНФРА-М, 2018. — 512 с. — (Среднее профессиональное образование). - Режим доступа: http://znanium.com/catalog/product/918098

**Материально-техническое обеспечение дисциплины**

1. Инструментальная среда разработки приложений MS Visual Studio.