

Министерство науки и высшего образования Российской Федерации

**Федеральное государственное бюджетное образовательное
учреждение высшего образования**

**«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)**

Факультет дистанционного обучения (ФДО)

А. В. Демаков, А. А. Квасников, С. П. Куксенко

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

**Методические указания
по выполнению лабораторных и самостоятельных работ
для обучающихся с применением дистанционных
образовательных технологий**

Томск 2022

Корректор: А. Н. Миронова

Демаков А. В. и др.

Объектно-ориентированное программирование : методические указания по выполнению лабораторных и самостоятельных работ для обучающихся с применением дистанционных образовательных технологий / А. В. Демаков, А. А. Квасников, С. П. Куксенко. – Томск : ФДО, ТУСУР, 2022. – 62 с.

© Демаков А. В., Квасников А. А.,
Куксенко С. П., 2022
© Оформление.
ФДО, ТУСУР, 2022

ОГЛАВЛЕНИЕ

Введение	4
1 Самостоятельная работа № 1 «Генераторы псевдослучайных чисел»	5
1.1 Цели и задачи работы	5
1.2 Краткие теоретические сведения и демонстрационные примеры	5
1.3 Задание на лабораторную работу	14
2 Лабораторная работа № 1 «Классы»	15
2.1 Цели и задачи работы	15
2.2 Теоретические сведения: ключевые темы	15
2.3 Программное и методическое обеспечение	15
2.4 Задание для выполнения работы	18
2.5 Порядок выполнения работы	18
2.6 Требования к оформлению отчета.....	19
2.7 Варианты индивидуальных заданий	20
3 Самостоятельная работа № 2 «Шаблоны».....	24
3.1 Цели и задачи работы	24
3.2 Краткие теоретические сведения и демонстрационные примеры	24
3.3 Задание на работу.....	31
4 Лабораторная работа № 2 «Наследование и виртуальные функции»	33
4.1 Цели и задачи работы	33
4.2 Теоретические сведения	33
4.3 Программное и методическое обеспечение	49
4.4 Задание для выполнения работы	49
4.5 Порядок выполнения работы	49
4.6 Требования к оформлению отчета.....	50
4.7 Варианты индивидуальных заданий	51
Список использованных источников	55
Приложение А (обязательное) Пользовательский проект	56

ВВЕДЕНИЕ

Я придумал термин «объектно-ориентированный»,
и я уверяю вас, что не имел в виду C++.

Алан Кей

Основными понятиями, лежащими в основе объектно-ориентированного программирования (ООП), являются класс и его объект. Так, согласно источнику [1], «объектно-ориентированное программирование – это методология программирования, которая основана на представлении программы в виде совокупности объектов, каждый из которых является реализацией определенного класса, а классы образуют иерархию на принципах наследуемости». Отсюда следует, что для понимания принципов ООП требуется уметь разрабатывать классы и работать с их объектами.

Поэтому общей целью данного пособия является получение навыков по разработке классов, созданию их объектов и оперированию ими. Для этого предусмотрены две лабораторные и две самостоятельные работы. Полученные в ходе их выполнения навыки и знания станут основой для дальнейшего изучения особенностей использования парадигмы ООП, широко применяемой на практике при разработке различного универсального и специализированного программного обеспечения для решения различных практических и научных задач.

После выполнения каждой лабораторной работы необходимо составить отчет и отправить его на проверку преподавателю. Если отчет удовлетворяет установленным требованиям, а задание выполнено надлежащим образом, то лабораторная работа идет в зачет дисциплины. Отчет возвращается на доработку до тех пор, пока все требования не будут выполнены. В случае, если есть основания полагать, что работа выполнена студентом не самостоятельно, преподаватель может задать дополнительные вопросы. В итоговой оценке учитывается качество выполнения задания, правильность и полнота ответов на дополнительные вопросы.

Самостоятельные работы отдельно не оцениваются, поэтому отчеты по ним не требуется отправлять на проверку.

1 САМОСТОЯТЕЛЬНАЯ РАБОТА № 1 «ГЕНЕРАТОРЫ ПСЕВДОСЛУЧАЙНЫХ ЧИСЕЛ»

Случайности не случайны.

М/ф «Кунг-фу панда»

1.1 Цели и задачи работы

Цель работы – изучить особенности использования генераторов псевдослучайных чисел в языке C++.

Для достижения поставленной цели необходимо решить следующие задачи:

- ознакомиться с представленными тестовыми примерами по генерации псевдослучайных чисел на языке C++ с разными видами распределений;
- выполнить модернизацию параметров шаблонов генераторов псевдослучайных чисел в представленных программных кодах;
- разработать программные коды с требуемыми видами распределений.

1.2 Краткие теоретические сведения и демонстрационные примеры

Генерация случайных чисел часто используется на практике, например в специализированных программах, играх и пр. Поэтому ознакомимся с тем, как генерировать случайные (рандомные) числа с использованием встроенных средств языка C++. Следует помнить, что для генерации случайных чисел используются специальные шаблонные функции, которые реализуют заложенный в них алгоритм, поэтому генерируемые случайные числа по сути не случайны, а псевдослучайны. При этом они могут иметь разные распределения.

В общем виде генерация псевдослучайных чисел основана на задании начального значения, с которым выполняются математические преобразования. Это позволяет конвертировать его в новое число, которое затем с использованием тех же преобразований дает еще одно новое число. Процесс продолжается до получения требуемой последовательности псевдослучайных чисел. Далее приведены демонстрационные примеры, разработанные для компилятора GDB online Debugger [2].

В язык C++ встроено несколько генераторов случайных чисел, как новых, так и «доставшихся по наследству» от языка C, в частности функции `rand()` и `srand()` (на некоторых компиляторах для их работы может потребоваться заголовочный файл `<cstdlib>`). Первая из них предназначена для задания начального значения для генерации псевдослучайных чисел (может отсутствовать), а вторая генерирует их от нуля до константы `RAND_MAX` (значение может различаться на разных компиляторах). Ниже приведен программный код их использования. Пример результата его работы приведен на рисунке 1.1.

```

1 | #include <iostream>
2 | using namespace std;
3 | int main(){
4 |     srand(1000); // srand(time(0));
5 |     for (auto i = 0; i < 15; i++){
6 |         cout << rand() << '\t';
7 |         if ((i + 1) % 3 == 0)
8 |             cout << endl;
9 |     }
10 |     cout << "RAND_MAX=" << RAND_MAX << endl;
11 |     return 0;
12 | }
```

766020790	1182770779	1333893513
173226398	1071903604	1702255141
2121871803	2124051570	983886268
1364009855	1991873138	779257283
1653856994	1570801147	147856433
RAND_MAX=2147483647		

Рис. 1.1 – Пример генерации псевдослучайных чисел
при использовании функций `srand()` и `rand()`

Если запустить этот код несколько раз, то результат (сгенерированные числа) будет одним и тем же. Чтобы получать разные результаты при каждом запуске, необходимо изменять значение аргумента функции `srand()` вручную. Однако можно автоматизировать это, например, используя функцию `time()` с нулевым значением аргумента (строка 4 кода). (На некоторых компиляторах для работы функции `time()` может потребоваться заголовочный файл `<ctime>`.) Функция возвращает число секунд, прошедшее с полуночи 01.01.1970 до момента её вызова. Поэтому при каждом запуске программы начальное значение для генерации псевдослучайных чисел будет разным.

Использование функции `rand()` прекрасно подходит при изучении программирования, но слабо применимо на практике, т. к. оно, как минимум, не позволяет явно генерировать числа с плавающей точкой (запятой). Кроме того, она не подходит, если нужно генерировать числа, превышающие `RAND_MAX`. Поэтому рассмотрим другие способы генерации псевдослучайных чисел, реализованные в заголовочном файле `<random>`. Их использование является предпочтительным по сравнению с устаревшей функцией `rand()`. Кроме того, они прекрасно сочетаются с контейнерами стандартной библиотеки шаблонов (STL) языка C++.

Генераторы псевдослучайных чисел, формирующие числа с равномерным распределением в указанном диапазоне, называют равномерными генераторами псевдослучайных чисел (РГСЧ, англ. URNG). Они характеризуются следующими свойствами:

- длина периода, т. е. число сгенерированных чисел, после которого эти числа начинают повторяться;
- производительность, т. е. затраты машинной памяти и времени, требуемые для генерации последовательности;
- стохастичность или качество, т. е. то, насколько генерируемая последовательность чисел близка к случайной.

Один из генераторов `<random>` – это вихрь Мерсенна (`std::mt19937`), который формирует случайные 32-битные целые беззнаковые числа (`unsigned int`). Существует также 64-битная версия (`std::mt19937_64`). Здесь использован наиболее известный алгоритм, названный MT19937, период которого составляет $2^{19937} - 1$. (Число Мерсенна – это число вида $M_n = 2^n - 1$, где n – натуральное число.)

Для демонстрации работы генератора применим следующий программный код, в котором сначала использовано начальное детерминированное значение, а затем недетерминированное значение, генерируемое с помощью объекта класса `random_device` (результаты равномерно распределены в диапазоне $[0, 2^{32})$). Пример результата его работы показан на рисунке 1.2. Имя генератора (использовано `gen`) может быть произвольным.

```

1 | #include <iostream>
2 | #include <random>
3 | using namespace std;
4 | int main(){
5 |     cout <<"-- deterministic 32-bit generator --" <<endl;
6 |     mt19937 gen(1000); // вихрь Мерсенна с заданием
    начального значения
7 |     for (auto i = 0; i < 15; i++){
8 |         cout << gen() << '\t'; // генерируемые 32-битные
    значения при каждом запуске будут ОДИНАКОВЫ
9 |         if ((i + 1) % 3 == 0)
10 |             cout << endl;
11 |     }
12 |     cout << "-- stochastic 32-bit generator --" << endl;
13 |     random_device rd; // задание случайного начального
    значения, недетерминированный (стохастический) генератор

```

```

14 | mt19937 gen_32(rd()); // вихрь Мерсенна со случайным
    | начальным значением
15 | for (auto i = 0; i < 15; i++){
16 |     cout << gen_32() << '\t'; // генерируемые
    | 32-битные значения при каждом запуске будут РАЗЛИЧНЫ
17 |         if ((i + 1) % 3 == 0)
18 |             cout << endl;
19 |     }
20 |     return 0;
21 | }

```

```

-- deterministic 32-bit generator --
2807145907      882709079      493951047
2621574848      4081433851     44058974
2070996316      1549632257     3747249597
3650674304      911961945      58396205
174846504       1478498153     1705937244
-- stochastic 32-bit generator --
3465764765      3092374054     362616658
1775880886      1642154037     2297831172
1838808206      2271903162     1682959543
3058204819      3320854527     2740241380
1994775372      2165420906     957931652

```

Рис. 1.2 – Пример генерации псевдослучайных чисел
при использовании `std::mt19937`

При задании начального значения с помощью объекта класса `random_device` генерируются псевдослучайные числа, которые различны при каждом запуске программы, но диапазон их разброса не контролируется. Для задания требуемого диапазона можно использовать однородное распределение (вероятность генерации всех чисел одинакова) с помощью шаблонных классов `std::uniform_int_distribution<>` (для генерации целых чисел) и `std::uniform_real_distribution<>` (для генерации чисел с плавающей точкой). Пример результата его работы показан на рисунке 1.3.

```

1  #include <iostream>
2  #include <random>
3  using namespace std;
4  int main(){
5      random_device rd;
6      cout<< "-- stochastic generator from a range of int
values --" <<endl;
7      uniform_int_distribution<int>dist(-15,15);
8      mt19937 gen_range(rd());
9      for (auto i = 0; i< 15; i++){
10         cout<<dist(gen_range) << '\t';
11         if ((i + 1) % 3 == 0)
12             cout<<endl;
13     }
14     cout<< "-- stochastic generator from a range of float
values --" <<endl;
15     uniform_real_distribution<double>dist_float(-
15.1,15.1);
16     mt19937 gen_range_float(rd());
17     for (auto i = 0; i< 15; i++){
18         cout<<dist_float(gen_range_float) << "    ";
19         if ((i + 1) % 3 == 0)
20             cout<<endl;
21     }
22     return 0;
23 }

```

```
-- stochastic generator from a range of int values --
-4      -3      1
-11     9      -6
7       -9     -12
-11     -5      9
-11     1      -6

-- stochastic generator from a range of float values --
8.45222  -1.86922  5.91572
-12.2666  13.3123  0.240502
2.44956  -9.88541  13.5928
-9.0893  -10.1424 -1.29102
-6.76597 -2.43064 -1.88044
```

Рис. 1.3 – Пример генерации псевдослучайных чисел
при использовании `std::mt19937` из заданного диапазона

Рассмотрим третий механизм равномерного распределения. Так, для преобразования сгенерированных чисел в значения с плавающей запятой (точкой) из их диапазона $[0, 1)$ используется шаблон `std::generate_canonical<>`, позволяющий задавать требуемые тип данных и число бит мантиисы. При преобразовании равномерное распределение сохраняется. Пример результата его использования приведен на рисунке 1.4.

```
1 | #include <iostream>
2 | #include <random>
3 | using namespace std;
4 | int main(){
5 |     random_device rd;
6 |     cout << "-- stochastic generator with generate
   | canonical --" << endl;
7 |     mt19937 gen(rd());
8 |     for (auto i = 0; i < 15; i++){
9 |         cout<<std::generate_canonical<float, 5>(gen)<<' ';
10 |         if ((i + 1) % 3 == 0)
```

```

11 |         cout << endl;
12 |     }
13 |     return 0;
14 | }

```

```

-- stochastic generator with generate canonical --
0.584187 0.564575 0.660237
0.146985 0.850514 0.502728
0.95594 0.0815856 0.51544
0.485214 0.673156 0.744271
0.109471 0.0914353 0.273847

```

Рис. 1.4 – Пример генерации псевдослучайных чисел

при использовании `std::mt19937` и `std::generate_canonical<>`

Для генерации чисел с дискретным и непрерывным распределениями в C++ используется много механизмов, включая биномиальное, Бернулли, геометрическое, Пуассона, Коши, Стьюдента и пр.

Например, шаблон `std::geometric_distribution<>` предназначен для задания геометрического распределения, а `std::poisson_distribution<>` – для пуассоновского. Оба работают с целыми неотрицательными числами. Шаблон `std::student_t_distribution<>` задает распределения Стьюдента для чисел с плавающей запятой (точкой). Следующий код демонстрирует использование этих распределений. Пример результата его работы показан на рисунке 1.5.

```

1 | #include <random>
2 | #include <iostream>
3 | using namespace std;
4 | int main(){
5 |     random_device rd;
6 |     mt19937 gen(rd());

```

```
7      cout << "-- stochastic generator with geometric
distribution --" << endl;
8      geometric_distribution<> distribGeo(0.5);
9      for (auto i = 0; i < 15; i++){
10         cout << distribGeo(gen) << " ";
11         if ((i + 1) % 3 == 0)
12            cout << endl;
13     }
14     cout << "-- stochastic generator with Poisson
distribution --" << endl;
15     poisson_distribution<int> distribPoisson(10);
16     for (auto i = 0; i < 15; i++){
17         cout << distribPoisson(gen) << " ";
18         if ((i + 1) % 3 == 0)
19            cout << endl;
20     }
21     cout << "-- stochastic generator with Student
distribution --" << endl;
22     student_t_distribution<double> distribStudent(5.0);
23     for (auto i = 0; i < 15; i++){
24         cout << distribStudent(gen) << " ";
25         if ((i + 1) % 3 == 0)
26            cout << endl;
27     }
28     return 0;
29 }
```

```
-- stochastic generator with geometric distribution --
0      0      1
0      0      7
0      6      2
0      0      1
1      3      1
-- stochastic generator with Poisson distribution --
7      13      9
5      8      7
11     8      11
9      8      11
8      18      11
-- stochastic generator with Student distribution --
-1.27187      0.456237      -0.676487
-3.9993       0.981968      1.55521
0.996671      1.28064       -1.34321
0.238776      -0.654921     -1.15184
-0.383618     -0.143547     0.513351
```

Рис. 1.5 – Пример генерации псевдослучайных чисел
при использовании `std::mt19937` при геометрическом,
пуассоновском и стьюдентовском распределениях

1.3 Задание на лабораторную работу

Для успешного выполнения задания следует:

1. Ознакомиться с программными кодами, приведенными в подразд. 1.2.
2. Модернизировать приведенные коды за счет изменения параметров шаблонов и проанализировать полученные результаты.
3. Разработать программные коды для генерации последовательностей псевдослучайных чисел при использовании распределений по следующим шаблонам:
 - `piecewise_linear_distribution<>` (кусочно-линейное);
 - `discrete_distribution<>` (дискретное);
 - `normal_distribution<>` (нормальное);
 - `lognormal_distribution<>` (логнормальное);
 - `bernoulli_distribution<>` (Бернулли);
 - `binomial_distribution<>` (биномиальное).

2 ЛАБОРАТОРНАЯ РАБОТА № 1 «КЛАССЫ»

Хе-хе, понятненько! Это должно быть выкопано,
положено в мешок, заперто в сундук.

М/ф «Клад»

2.1 Цели и задачи работы

Цель работы – изучить особенности разработки классов на языке C++, а также получить навыки по созданию и работе с их объектами.

Для достижения поставленной цели необходимо решить следующие главные задачи:

- ознакомиться с теоретическими сведениями;
- программно реализовать пользовательский класс согласно заданию, включающему общую (разработать класс, содержащий поля и методы) и индивидуальную (дополнительный метод класса) части;
- выполнить тестирование программного кода;
- оформить отчет по проделанной работе.

2.2 Теоретические сведения: ключевые темы

Для успешного выполнения задания рекомендуется предварительно ознакомиться с материалами п. 3.3.1–3.3.7 (раздел 3. ООП: инкапсуляция) учебного пособия по дисциплине [3] и просмотреть небольшие видеофайлы:



[Что такое класс. Что такое объект класса.](#)

[Сеттеры и геттеры, указатель this.](#)

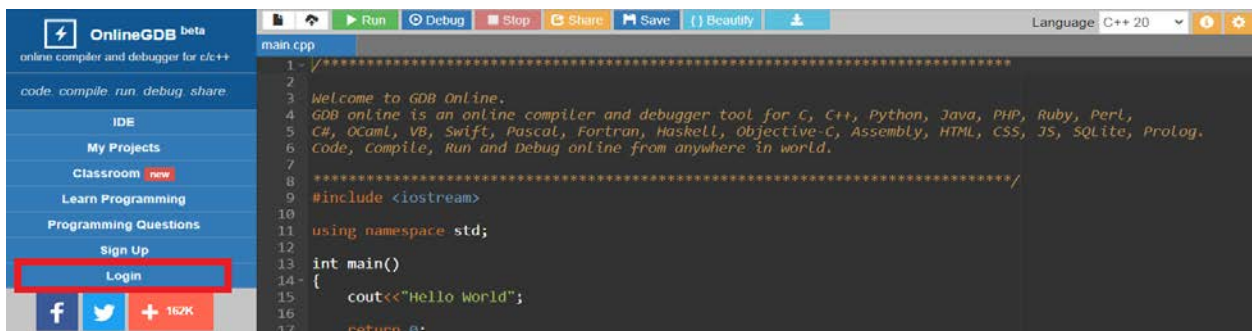
[Конструкторы и деструкторы.](#)

2.3 Программное и методическое обеспечение

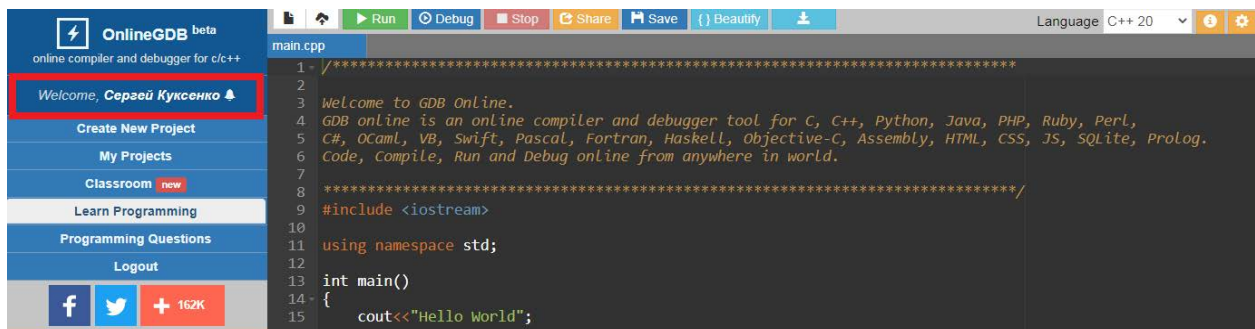
Работу можно выполнить в интегрированных средах разработки Microsoft Visual Studio или Qt Creator (инструкции по их установке и настройке, а также

созданию проектов в них приведены в приложениях А, Б, Д и Е учебного пособия по дисциплине [3]) или используя компилятор GDB online Debugger [2] (для работы требуется браузер и выход в сеть Интернет).

Кратко рассмотрим особенности создания пользовательского проекта в компиляторе GDB online Debugger. Для этого на главной странице компилятора надо пройти процедуру регистрации или, если она уже пройдена ранее, выполнить вход в аккаунт. Для этого в левой части экрана надо выбрать Login (рис. 2.1, а). Затем надо выбрать требуемый режим входа. Для этого можно выбрать имеющиеся аккаунты в сервисах Google, Facebook, GitHub или зарегистрироваться по адресу электронной почты (рис. 2.1, б).



а



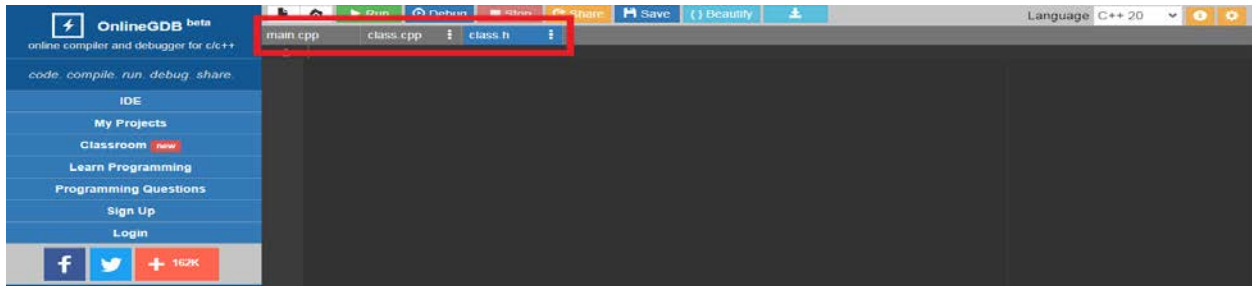
б

Рис. 2.1 – GDB online Debugger: вид до (а) и после (б) входа в аккаунт

По умолчанию пользовательский проект состоит из одного файла `main.cpp`, в который помещена функция `main()`. Для создания дополнительного файла (для разделения определения и реализации классов) надо в верхней части окна нажать на иконку **New File** (или **Ctrl+M**), ввести требуемое имя и нажать **Save** (рис. 2.2).



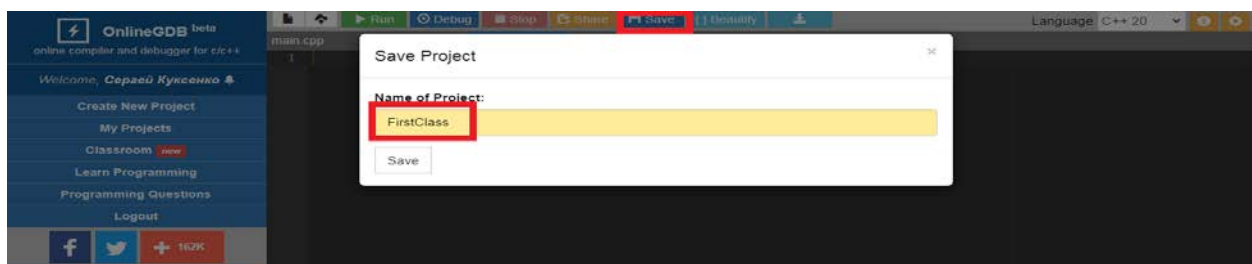
а



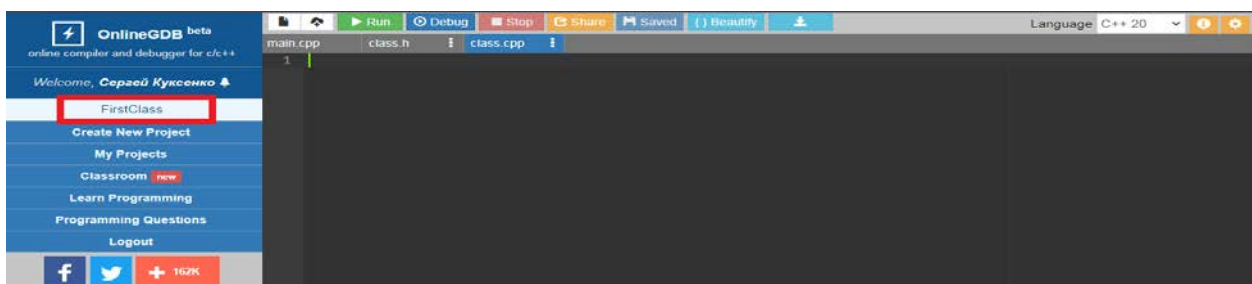
б

Рис. 2.2 – GDB online Debugger: добавление нового файла (а)
и вид окна с пользовательским проектом, состоящим из 3 файлов (б)

Для сохранения файлов в пользовательском проекте следует нажать на иконку **Save** (Ctrl+S), расположенную по центру в верхней части окна, и ввести требуемое имя проекта (рис. 2.3, а). После описанных действий главное окно примет вид, представленный на рисунке 2.3, б.



а



б

Рис. 2.3 – GDB online Debugger: добавление имени проекта (а)
и вид окна с успешным сохраненным проектом (б)

Для возврата к редактированию проекта после повторного входа в пользовательский аккаунт используется меню в левой части экрана `My Projects`.

2.4 Задание для выполнения работы

В ходе работы необходимо, *не используя контейнеры библиотеки шаблонов STL*, разработать класс `Array`, содержащий:

- конструктор (задает начальные значения полям класса);
- деструктор (освобождает память, которая выделялась под массив);
- метод, резервирующий память под массив, состоящий из N элементов, где N – это аргумент метода;
- метод, задающий значение k -го элемента массива, где индекс k и значение элемента массива являются аргументами метода;
- метод, заполняющий массив псевдослучайными числами с распределением согласно индивидуальному заданию;
- метод, выводящий элементы массива на экран (консольное окно);
- метод, выводящий на экран текущий размер массива, выраженный в числе его элементов и в байтах, для его хранения;
- дополнительный метод (варианты см. в таблице 2.1);
- поле, хранящее размер массива;
- поле, хранящее указатель на массив.

Пример *определения* (`array.cpp`), *реализации* (`array.h`) и использования (`main.cpp`) класса `array` приведен в приложении А.

2.5 Порядок выполнения работы

При выполнении работы необходимо:

1. Создать проект (подразд. 2.3) с именем, состоящим из имени студента и номера учебной группы (например, `Sidorov_169_1`) и содержащим файлы с программным кодом из приложения А. (Объявление класса поместить в заголовочный файл `*.h`, а его реализацию – в исполняемый `*.cpp`. Имена этих файлов

должны совпадать с именем разрабатываемого класса. Функция `main()`, создание объектов класса и работа с ними должны быть помещены в файл `main.cpp`.)

2. Выполнить анализ работы программного кода из созданного проекта.

3. Выполнить модернизацию программного кода созданного проекта для того, чтобы его элементы были именованы согласно рекомендациям из приложения В учебного пособия по дисциплине [3].

4. Модернизировать программный код класса, чтобы он содержал все методы, указанные в подразд. 2.4.

5. При реализации методов класса и программы в целом необходимо предусмотреть обработку ошибочных ситуаций: неверный выбор режима работы программы; индекс k выходит за рамки массива; индекс k меньше нуля или больше размера массива; размер массива меньше или равен нулю и др. Для этого необходимо реализовать обработку исключений, используя оператор `throw` и блоки `try` и `catch`.

6. Выполнить тестирование работы созданного проекта. Для этого в функции `main()` создать объект разработанного класса и с его помощью продемонстрировать работу всех методов класса. Вызов каждого метода (кроме конструктора и деструктора) организовать с помощью считывания данных из консоли и вывода в консоль. При этом должно быть выполнено тестирование всех ветвлений работы методов класса как при корректном, так и при некорректном задании требуемых параметров (аргументов методов).

7. Оформить отчет по проделанной работе согласно требованиям из подразд. 2.6.

2.6 Требования к оформлению отчета

Структура отчета должна включать:

- титульный лист;
- цель работы;
- основную часть (описательная часть, пояснения, результаты тестирования);
- выводы по работе;

- список использованной литературы;
- приложения (3 шт.), содержащие отдельные листинги файлов с программным кодом (3 файла) созданного проекта.

При оформлении текстовой части отчета следует руководствоваться требованиями Образовательного стандарта ТУСУР 01-2021 «Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления» (приказ ректора от 25.11.2021) (<https://regulations.tusur.ru/documents/70>).

Отчет по работе должен включать:

- архив (RAR или ZIP) с программной частью лабораторной работы. В архиве должны находиться все исходные файлы, включая файлы проектов используемой среды разработки, а также файлы, необходимые для запуска программы (исполняемый файл, файлы с входными данными и т. п.). Программный код должен быть построчно комментирован;
- документ с отчетом по лабораторной работе в формате DOC, DOCX, RTF или PDF;
- рецензия на предыдущий вариант работы, если она сдается повторно после исправления замечаний.

В случае, если есть основания полагать, что работа выполнена студентом не самостоятельно, преподаватель может задать дополнительные вопросы. В итоговой оценке учитывается качество выполнения задания, правильность и полнота ответов на дополнительные вопросы.

2.7 Варианты индивидуальных заданий

Работа выполняется согласно варианту индивидуального задания (таблица 2.1), выбор которого осуществляется по следующей формуле:

$$V = NK \div 100,$$

где V – искомый номер варианта;

N – общее количество вариантов;

K – код варианта (индивидуальный код студента);

div – целочисленное деление.

При $V = 0$ выбирается максимальный вариант.

Параметры, не указанные в таблице 2.1, но необходимые для работы генераторов псевдослучайных чисел, выбираются на усмотрение обучающегося.

Таблица 2.1 – Варианты индивидуальных заданий

Вариант	Задание
1	Разработать класс <code>array_delete_first_zero</code> для работы с массивом динамической длины. Тип данных – <code>int</code> . Для генерации псевдослучайных чисел использовать 32-битный алгоритм MT19937 и начальное значение 1234. Дополнительный метод: удаление из массива первого найденного нулевого элемента со сдвигом элементов (уменьшение размера массива)
2	Разработать класс <code>array_sort</code> для работы с массивами динамической длины. Тип данных – <code>int</code> . Для генерации псевдослучайных чисел использовать 32-битный алгоритм MT19937 и случайное начальное значение. Дополнительный метод: сортировка элементов массива
3	Разработать класс <code>array_delete</code> для работы с массивами динамической длины. Тип данных – <code>int</code> . Для генерации псевдослучайных чисел использовать 64-битный алгоритм MT19937 и начальное значение 2345. Дополнительный метод: удаление элемента с номером n (n – аргумент метода, заданный пользователем)
4	Разработать класс <code>array_last</code> для работы с массивами динамической длины. Тип данных – <code>int</code> . Для генерации псевдослучайных чисел использовать 64-битный алгоритм MT19937 и случайное начальное значение. Дополнительный метод: вывод последних n элементов массива (n – аргумент метода, заданный пользователем)
5	Разработать класс <code>array_double_directed</code> для работы с массивами динамической длины. Тип данных – <code>int</code> . Для генерации псевдослучайных чисел использовать 32-битный алгоритм MT19937, случайное начальное значение и равномерное распределение для диапазона $(-13, 10)$. Дополнительный метод: вывод массива на экран в обратном порядке
6	Разработать класс <code>array_remove_if_odd</code> для работы с массивами динамической длины. Тип данных – <code>float</code> . Для генерации псевдослучайных чисел использовать 32-битный алгоритм MT19937, случайное начальное значение и равномерное распределение для диапазона $(-6.6, 6.6)$. Дополнительный метод: удаление элементов с четными индексами со сдвигом оставшихся элементов (уменьшение размера массива)

Продолжение таблицы 2.1

Вариант	Задание
7	Разработать класс <code>array_remove_if_odd_value</code> для работы с массивами динамической длины. Тип данных – <code>float</code> . Для генерации псевдослучайных чисел использовать 32-битный алгоритм MT19937, случайное начальное значение и равномерное распределение для диапазона $[0, 1)$. «Дополнительный метод: удаление элементов с четными индексами со сдвигом оставшихся элементов (уменьшение размера массива)»
8	Разработать класс <code>array_remove_if_not_odd</code> для работы с массивами динамической длины. Тип данных – <code>int</code> . Для генерации псевдослучайных чисел использовать 32-битный алгоритм MT19937, случайное начальное значение и геометрическое распределение. Дополнительный метод: удаление элементов с нечетными индексами со сдвигом оставшихся элементов (уменьшение размера массива)
9	Разработать класс <code>array_remove_if_not_odd_value</code> для работы с массивами динамической длины. Тип данных – <code>int</code> . Для генерации псевдослучайных чисел использовать 32-битный алгоритм MT19937, случайное начальное значение и распределение Пуассона. «Дополнительный метод: удаление элементов с четными индексами со сдвигом оставшихся элементов (уменьшение размера массива)»
10	Разработать класс <code>array_remove_every_N</code> для работы с массивами динамической длины. Тип данных – <code>float</code> . Для генерации псевдослучайных чисел использовать 32-битный алгоритм MT19937, случайное начальное значение и распределение Стюдента. Дополнительный метод: удаление каждого n -го элемента массива (n – аргумент метода, заданный пользователем) со сдвигом оставшихся элементов (уменьшение размера массива)
11	Разработать класс <code>array_shift_left</code> для работы с массивами динамической длины. Тип данных – <code>int</code> . Для генерации псевдослучайных чисел использовать 32-битный алгоритм MT19937 и начальное значение 4321. Дополнительный метод: циклический сдвиг элементов массива влево на n позиций (n – аргумент метода, заданный пользователем). Пример: было <code>[1 10 3 5 7 2 4]</code> и при $n = 3$ стало <code>[5 7 2 4 1 10 3]</code>
12	Разработать класс <code>array_shift_right</code> для работы с массивами динамической длины. Тип данных – <code>int</code> . Для генерации псевдослучайных чисел использовать 32-битный алгоритм MT19937 и случайное начальное значение. Дополнительный метод: циклический сдвиг элементов массива вправо на n позиций (n – аргумент метода, заданный пользователем). Пример: было <code>[1 10 3 5 7 2 4]</code> и при $n = 3$ стало <code>[7 2 4 1 10 3 5]</code>

Окончание таблицы 2.1

Вариант	Задание
13	Разработать класс <code>array_delete_max_positive_items</code> для работы с массивами динамической длины. Тип данных – <code>float</code> . Для генерации псевдослучайных чисел использовать 32-битный алгоритм MT19937, случайное начальное значение и равномерное распределение для диапазона $(-13.13, 13.13)$. Дополнительный метод: удаление из массива максимального положительного элемента со сдвигом оставшихся элементов (уменьшение размера массива)
14	Разработать класс <code>array_delete_max_negative_items</code> для работы с массивами динамической длины. Тип данных – <code>float</code> . Для генерации псевдослучайных чисел использовать 32-битный алгоритм MT19937, случайное начальное значение и распределение Стьюдента. Дополнительный метод: удаление из массива максимального отрицательного элемента со сдвигом оставшихся элементов (уменьшение размера массива)
15	Разработать класс <code>array_delete_min_positive_items</code> для работы с массивами динамической длины. Для генерации псевдослучайных чисел использовать 32-битный алгоритм MT19937, случайное начальное значение и равномерное распределение для диапазона $(-15, 15)$. Дополнительный метод: удаление из массива минимального положительного элемента со сдвигом оставшихся элементов (уменьшение размера массива)

3 САМОСТОЯТЕЛЬНАЯ РАБОТА № 2 «ШАБЛОНЫ»

Чужие слабости надо уважать.
Х/ф «Жмурки»

3.1 Цели и задачи работы

Цель работы – изучить особенности разработки и использования шаблонов функций в языке C++.

Для достижения поставленной цели необходимо решить следующие задачи:

- ознакомиться с представленными тестовыми примерами по разработке шаблонов функций в языке C++;
- разработать программные коды для реализации предписанных заданием шаблонных функций.

3.2 Краткие теоретические сведения и демонстрационные примеры

3.2.1 Механизм шаблонов

Шаблоны позволяют давать обобщенные определения функций и классов, поэтому их часто называют «шаблонные функции» и «шаблонные классы». Кроме того, используются термины «параметризованные функции» и «параметризованные классы».

Шаблон функции представляет собой обобщенное определение функции, из которого компилятор автоматически создает представителя функции для заданного пользователем типа (или типов) данных. Когда компилятор по шаблону функции создает ее конкретную реализацию, то говорят о *порожденной функции*.

При создании шаблона функции используется ключевое слово `template`, за которым следуют один или несколько параметров (обобщенные имена типов данных, используемые внутри шаблона), заключенных в угловые скобки и разделенные между собой запятыми. Вместо ключевого слова `class` может также использоваться ключевое слово `typename` (однако это справедливо только для

шаблона функции, но не шаблона класса, поэтому использование слова `class` более универсально).

Синтаксис определения шаблонной функции (квадратные скобки сигнализируют о том, что заключенные в них параметры могут отсутствовать) имеет следующий вид:

<i>Заголовок функции</i>	template <class T1 [, . . . , class Tn]>
	return_type func_name (arguments)
	{
<i>Тело функции</i>	}

Параметры шаблона, следующие за ключевыми словами `class` (или `typename`), называют *параметризованными типами* (T1, T2, ...), имена которых могут быть произвольными. Они информируют компилятор о том, что некоторый (пока что неизвестный) тип данных используется в шаблоне в качестве параметра (в момент вызова шаблона на их место подставляются требуемые типы).

Более общее определение шаблона позволяет указывать параметры, которые не относятся к параметризованным типам, их иногда называют идентификаторы (I1, ..., Ik). Они могут быть указаны в любом месте в угловых скобках, но рекомендуется указывать их после параметров шаблонного типа.

Синтаксис общего определения шаблона функции:

<i>Заголовок функции</i>	template <class T1 [, . . . , class Tn, type I1, ... , type Ik]>
	return_type funcn_name (arguments)
	{
<i>Тело функции</i>	}

В результате шаблонные функции можно вызывать как обычные функции, например, как показано в следующем программном коде. Результат его работы приведен на рисунке 3.1.

```

1  #include <iostream>
2  using namespace std;
3  template <class T>    // Шаблон для вычисления квадрата
4  T Pow(T x){
5      return x*x;
6  }
7  template <class T>    // Шаблон перестановки элементов
8  T* Swap(int I1, int I2, T* t) { // здесь параметры
    // намеренно расположены в нерекомендуемом порядке для
    // демонстрации такой функциональности
9      T tmp = t[I1];
10     t[I1] = t[I2];
11     t[I2] = tmp;
12     return t;
13 }
14 int main( ){
15     int n=10, i = 2, j = 5;
16     double d = 10.21;
17     float arr[5] = {1.1, 2.2, 3.3, 4.4, 5.5};
18     string ss = "Template";
19     char* aa = &ss[0];
20     cout<< "The value of the n = " << n << endl << "The
    square of the n = " << Pow(n) << endl<< "The value of d
    = " << d << endl << "The square of d = " << Pow(d) <<
    endl << "The initial string = " << aa << endl << "The
    modified string = " << Swap(i, j, aa) << endl << "The
    initial array = ";
21     for(auto i: arr)
22         cout << i << " ";
23     cout << endl << "The modified array = ";

```

```

24 | float * parr = Swap(0,4,arr);
25 | for(auto i: arr)
26 |     cout << i << " ";
27 | return 0;
28 | }

```

```

The value of the n = 10
The square of the n = 100
The value of d = 10.21
The square of d = 104.244
The initial string = Template
The modified string = Template
The initial array = 1.1 2.2 3.3 4.4 5.5
The modified array = 5.5 2.2 3.3 4.4 1.1

```

Рис. 3.1 – Результат работы шаблонных функций

В приведенном примере вызовы шаблонной функции `Sqr()` и `Sqr()` для `n` и `d` приводят к построению двух *порожденных функций* для параметризованных типов, в данном случае это `int` и `double` соответственно. Аналогичное происходит и с шаблонной функцией `Swap()` с параметризованными типами `char*` и `float*`. Поэтому процесс построения компилятором порожденной функции иногда называют *конкретизацией шаблонной функции*.

Стоит обратить внимание на то, что в приведенном примере при вызове шаблонных функций явно не были указаны параметризованные типы, поскольку компилятор по типу данных аргументов самостоятельно «выбрал» требуемые реализации. Однако это не всегда возможно, поэтому рекомендуется явно указывать параметризованные типы при вызове шаблонных функций. В рассмотренном примере это соответствует `Pow<int>(n)`, `Pow<double>(d)`, `Swap<char>(i, j, aa)` и `Swap<char>(i, j, aa)`.

Как и для обычных функций, можно создать прототип шаблона функции в виде его предварительного объявления. Например:

```
«template < class T> T* Swap(int I1, int I2 , T* t);».
```

При этом рекомендуется и объявление, и реализацию шаблонной функции помещать в заголовочный файл.

Важно помнить, что каждый параметризованный тип, указанный в списке параметров, должен использоваться в теле функции. Например, объявление

```
«template <class T1, class T2> T2 func (T2);»
```

приведет к ошибке во время компиляции, т. к. тип `T1` не используется в теле функции. (Подробное описание шаблонов дано в учебном пособии по дисциплине [3].)

3.2.2 Вариативные шаблоны

До этого момента были рассмотрены шаблоны, принимающие строго определенное число параметров, однако иногда требуется, чтобы это число могло измениться в зависимости от контекста выполнения программы. Для этого используются вариативные шаблоны.

Так, вариативный шаблон (англ. *variadic template*) – это шаблон функции (или класса), который поддерживает *произвольное* число параметров функции. Набор таких параметров называется пакетом параметров (*parameter pack*).

Есть два вида пакетов параметров: пакет параметров шаблона (*template parameter pack*), представляющий любое число параметров шаблона, и пакет параметров функции (*function parameter pack*), представляющий любое число параметров функции.

Для указания, что шаблону или функции представлен пакет параметров, используется оператор многоточие (`...`). При этом он используется в двух контекстах. Так, если он помещен слева от имени параметра, то он обозначает *пакет параметров*, а справа – *распаковку пакета параметров* на отдельные имена. Поэтому положение многоточия очень важно:

```
1 | template <class ... Types>
2 | void func1(std::vector<Types ...> v1); // v1 НЕ является
   | пакетом параметров функции
```

```

3 | template <class... Types> // v2 ЯВЛЯЕТСЯ пакетом
   | параметров функции
4 | void func2(std::vector<Types>... v2);
5 | template <class... T> // v3 ЯВЛЯЕТСЯ пакетом параметров
   | функции
6 | void func3(const T&... v3);

```

Здесь, в записях `vector<Types>... v2` и `const T&... v3`, многоточия обозначают пакеты параметров `v2` и `v3` соответственно.

Одна из сложностей использования вариативных шаблонов заключается в работе с его параметрами. Привычные механизмы индексации (`[ ]`) здесь не работают. Поэтому рассмотрим распаковку пакета с использованием рекурсии.

Так, вариативные шаблоны функций (как и аналогичные вариативные шаблоны классов) могут устанавливать требование о том, что должен быть передан по крайней мере один параметр. По сути, нужно подсказать компилятору, что первый параметр функции – это тип данных, а остальная часть – пакет. Их определение имеет следующий вид:

```

template <class T, class ... U>
return_type func_name(const T& first, const U& ... rest);

```

Здесь параметр `first` является обязательным, а `rest`, обобщающий возможные остальные параметры, нет. Отметим, что в функции могут передаваться большие объекты, поэтому рекомендуется ее аргументы передавать по ссылке (`T&`) или, что еще лучше, по константной ссылке (`const T&` и `const U&`).

Следующий программный код демонстрирует работы вариативного шаблона функции. Для его работы требуется сначала переопределить функцию с одним параметром. (Если по логике программы надо, чтобы функция также работала и без параметров, то необходимо сделать еще одну перегрузку функции под этот случай.) Результат работы приведен на рисунке 3.2.

```

1  #include <iostream>
2  using namespace std;
3  template <class T>
4  void func(const T& val){ // функция для обработки случая
   с одним параметром
5      cout << val << endl;
6  }
7  template<class T, class ... U> // U – это пакет
   параметров шаблона
8  void func(const T& first, const U& ... rest){ // rest -
   это пакет параметров функции
9      cout << first << ", ";
10     func(rest...); // распаковка пакета параметров за
   счет рекурсивных вызовов функции с одним параметром
11 }
12 int main(){
13     func(100);
14     func("first", 1);
15     func(2 , "second", 2);
16     func("third", 3);
17 }

```

```

first, 1
2, second, 2
third, 3

```

Рис. 3.2 – Результат работы вариативной шаблонной функции

Стоит отметить, что в вариативных шаблонах используется оператор `sizeof...()`, который отличается от привычного оператора `sizeof()`.

```

1 | template<typename ... T>
2 | void func(T ... arg) {
3 |     cout << sizeof...(T) << endl; // число параметров
   | пакета шаблона
4 |     cout << sizeof...(args) << endl; // число параметров
   | пакета функции
5 | }

```

3.2.3 Недостатки шаблонов

Шаблоны предоставляют определенные выгоды при программировании, связанные с широкой применимостью кода и легким его сопровождением. Этот механизм позволяет решать те же задачи, для которых используется полиморфизм. С другой стороны, в отличие от макросов, они позволяют обеспечить безопасное использование типов данных.

Однако с использованием шаблонов связаны и некоторые недостатки:

- программа содержит полный код для всех порожденных представителей шаблонного класса или функций;
- не для всех типов данных предусмотренная реализация класса или функции оптимальна.

Преодоление второго недостатка возможно с помощью специализации шаблонов или использования концептов (начиная с C++ 20). Отметим, что специализация шаблонов подробно рассмотрена в учебном пособии по дисциплине [3], а изучение концептов отводится на самостоятельное изучение.

3.3 Задание на работу

Разработать шаблонную функцию и продемонстрировать корректность ее работы для типов данных `int`, `double` и `char`. При необходимости использовать вариативные шаблоны. Разрабатываемая функция должна выполнять с заданной матрицей (двумерного массива) произвольного размера совместно два

из нижеследующих действий (параметры, не указанные в задании, но необходимые для функции, выбираются на усмотрение обучающегося):

- транспонировать матрицы;
- менять местами заданные i -ю строку и j -й столбец;
- определять, существуют ли в матрице повторяющиеся строки;
- определять элемент, который встречается в массиве максимальное число раз;
- упорядочивать строки матрицы по возрастанию значения первого элемента в строке;
- переставлять i -ю и j -ю строки;
- вычислять максимальное значение элемента в массиве;
- менять элементы главной и побочной диагоналей местами;
- заменять все элементы, меньшие введенного значения, на это значение;
- проверять матрицу на диагональное преобладание.

4 ЛАБОРАТОРНАЯ РАБОТА № 2 «НАСЛЕДОВАНИЕ И ВИРТУАЛЬНЫЕ ФУНКЦИИ»

Нам с вами надо работать конструктивнее. Надо, так сказать, начинать уже поэтапно, планомерно ломать вот эти, устоявшиеся стереотипы. А нам их с вами досталось, так сказать, в наследство... не мне вам не говорить.

Х/ф «Какраки»

4.1 Цели и задачи работы

Цель работы – приобрести практические навыки по использованию механизмов наследования и виртуальных функций при разработке объектно-ориентированных программ.

Для достижения поставленной цели надо решить следующие главные задачи:

- ознакомиться с теоретическими сведениями;
- программно реализовать иерархию классов;
- выполнить тестирование программного кода;
- оформить отчет по проделанной работе.

4.2 Теоретические сведения

4.2.1 Простое наследование

Язык C++ позволяет классу наследовать члены (поля и методы) одного или нескольких других классов. При этом новый класс называют производным (или классом-потомком). Класс, члены которого наследуются, называется базовым (родительским классом или классом-предком) для своего производного класса.

Наследование дает возможность абстрагировать некоторые общие черты поведения классов в одном базовом классе. Производные классы, наследуя это общее поведение, могут его несколько видоизменять, переопределяя некоторые методы базового класса, или дополнять, вводя новые поля и методы. В результате

определение производного класса может быть значительно сокращено, т. к. нужно определить только эти новые члены, отличающие его от производных классов.

Если у производного класса имеется всего один базовый класс, то говорят о простом (или одиночном) наследовании. Синтаксис объявления производного класса имеет следующий вид:

```

1 | class <имя класса> : [<спецификатор доступа>] <имя
   | базового класса> {
2 |     <тело класса>
3 | };

```

Спецификатор доступа – это одно из ключевых слов `private`, `protected` или `public`. Он не является обязательным и по умолчанию принимает значение `private` для классов (`public` для структур). Рассмотрим пример:

```

1 | #include <iostream>
2 | using namespace std;
3 | class Base{ // базовый класс
4 |     public:
5 |         int SetX(int _x=0){
6 |             return x=_x;
7 |         }
8 |         int GetX( ) {
9 |             return x;
10 |        }
11 |     protected: // обозначает, что
12 |         int x; // данное поле доступно данному классу и
   |               // всем его потомкам
13 | };
14 | class Derived : public Base{ // производный класс

```

```

15  // требуемый код
16 };
17 int main(){
18     int x;
19     Derived ob; // создание объекта производного класса
20     x = ob.GetX( ); // обращение через этот объект к
    // полю и методу базового класса
21     return 0;
22 }

```

Спецификатор доступа при наследовании определяет уровень доступа к членам базового класса, который получают члены производного класса. В таблице 4.1 описаны возможные варианты наследуемого доступа. Из таблицы следует, что при наследовании, как правило, происходит понижение доступа к элементам базового класса.

Таблица 4.1 – Спецификаторы доступа в рамках иерархии наследования

Доступ в базовом классе	Спецификатор наследуемого доступа	Доступ в производном классе
public protected private	public	public protected Нет доступа
public protected private	protected	protected protected Нет доступа
public protected private	private	private private Нет доступа

Стоит помнить, что не все члены базового класса могут наследоваться. Так, не подлежат наследованию: конструкторы, в том числе конструкторы копирования; деструкторы; перегруженные операторы присваивания; дружественные функции.

4.2.2 Переопределение методов

В производном классе обычно добавляются новые поля и методы, отсутствующие в базовом классе. Кроме того, можно переопределять методы базового класса. При этом очевидно, что определения таких методов в базовом и производном классах должны совпадать, как это сделано в следующем программном коде на примере геттеров.

```

1  #include <iostream>
2  using namespace std;
3  class MyInt{
4      public:
5          MyInt(int _x) {x = _x;}
6          MyInt( ){x = 0;}
7          int GetX( ){return x;} // геттер базового класса
8          void SetX(int _x) {x = _x;} // сеттер базового класса
9      protected:
10         int x;
11 };
12 class IncMyInt : public MyInt{
13     public:
14         IncMyInt(int _x) : MyInt(_x) {}
15         int GetX( ) {return ++x;} // переопределенный геттер
16         // базового класса
17         void SetX(int _x) {x = _x+900;} // переопределенный
18         // сеттер базового класса
19         void ShowX( ) {cout << GetX() << ' ' ;}
20 };
21 int main( ){
22     IncMyInt* ptr;
23     ptr = new IncMyInt(10); // создание объекта в куче

```

```

22     ptr->ShowX( );
23     delete ptr; // освобождение памяти
24     return 0;
25 }

```

Здесь создается объект класса `IncMyInt` при помощи конструктора (`x = 10`). Далее метод `ShowX( )` этого объекта использует для получения значения поля `x` переопределенный вариант метода (геттера) базового класса `GetX( )`. В результате в консоль будет выведено инкрементированное значение поля `x` (`x = 11`).

Для того чтобы вызывать метод базового класса, а не его переопределенный вариант, используется оператор разрешения области видимости (`::`):

«<имя_класса>::*имя_метода*>».

Это дает возможность компилятору «видеть» за пределами текущей области видимости. Так, переопределенный метод может вызывать соответствующий метод базового класса, а затем выполнять некоторую дополнительную работу (или наоборот). Например, в предыдущем примере можно было переопределить метод `GetX( )` с помощью следующего кода:

```

1  int IncMyInt::GetX(){// метод производного класса,
2  являющийся переопределенным методом базового класса
3      int z;
4      z = MyInt::GetX(); // вызов метода базового класса
5      return ++z;
6  }

```

4.2.3 Конструкторы и деструкторы

Как уже упоминалось ранее, конструкторы не наследуются, поэтому производный класс должен объявить свой конструктор или предоставить возможность компилятору генерировать конструктор по умолчанию. Поскольку производный класс должен унаследовать все члены базового класса, то при его определении должна быть обеспечена инициализация унаследованных полей, причем она должна быть выполнена до инициализации полей производного класса, так как последние могут использовать значения первых. Поэтому при определении конструктора производного класса применяется следующая конструкция:

```
«<имя класса> ([<список аргументов>]): <имя базового
класса> ([<список аргументов>]) { // тело конструктора про-
изводного класса }».
```

Так, часть аргументов, переданных конструктору производного класса, используется в качестве аргументов конструктора базового класса. Затем в теле конструктора производного класса выполняется инициализация его полей. В приведенном выше примере указанная конструкция использована для определения конструктора класса IncMyInt:

```
«IncMyInt (int _x): MyInt (_x) {}».
```

Здесь тело конструктора оставлено пустым, так как класс IncMyInt не имеет собственных полей. При этом аргументы конструктора этого производного класса переданы конструктору базового класса для инициализации его поля X.

В случае если реализованный конструктор производного класса не имеет аргументов (конструктор по умолчанию), то при создании его объекта автоматически вызывается конструктор базового класса. После того как объект создан, конструктор базового класса становится недоступным.

В отношении деструкторов производных классов также действуют определенные правила. Так, деструктор производного класса должен вызываться раньше деструктора базового класса (иначе деструктор базового класса мог бы разрушить данные полей, которые используются и в производном классе). После

того как деструктор производного класса выполнит свою часть работы по уничтожению объекта, вызывается деструктор базового класса. Причем вся работа по организации соответствующего вызова возлагается на компилятор.

4.2.4 Виртуальные методы

Работа с объектами класса чаще всего производится через указатели. Указателю на базовый класс («`MyInt *p;`») можно присвоить адрес объекта любого производного от него класса («`p = new IncMyInt;`»).

Продemonстрируем сказанное на примере следующего кода, в котором использованы описанные выше базовый класс `MyInt` и производный класс `IncMyInt`, не имеющие виртуальных методов. Результат работы этого программного кода приведен на рисунке 4.1.

```

1 | int main( ){
2 |     MyInt*p; // указатель на базовый класс
3 |     p = new MyInt(0);
4 |     cout<<p->GetX() <<endl; // метод базового класса,
   |     здесь результат 0
5 |     p->SetX(111); // метод базового класса
6 |     cout<<p->GetX() <<endl; // метод базового класса,
   |     здесь результат 111
7 |     p = new IncMyInt(100); // здесь указатель ссылается
   |     на производный класс
8 |     cout<<p->GetX() <<endl; // метод базового класса,
   |     здесь результат 100
9 |     p->SetX(222);
10 |    cout<<p->GetX() <<endl; // метод базового класса,
   |    здесь результат 222
11 |    static_cast<IncMyInt*>(p)->SetX(33333333); //
   |    (в стиле языка C - (IncMyInt* p)->SetX(33333333);) метод
   |    ПРОИЗВОДНОГО класса

```

```

12      cout<<p->GetX() <<endl; // метод базового класса,
      здесь результат 33334233
13      cout<<static_cast<IncMyInt*>(p)->GetX() <<endl;
      //метод ПРОИЗВОДНОГО класса, здесь результат 33334234
14      return 0;
15  }

```

Дадим некоторые пояснения по представленному коду. Для некоторого класса вызов методов через указатель происходит в соответствии с типом, использованным при создании этого указателя, а не фактическим (присвоенным) типом, поэтому при выполнении оператора разыменования (строки 4–6, а также 8–10 и 12) вызываются методы класса `MyInt`, а не класса `IncMyInt`. Это связано с тем, что ссылки на методы устанавливаются во время компоновки программы. Такой процесс основан на механизме раннего связывания. Тогда, чтобы вызвать методы класса `IncMyInt`, следует использовать явное приведение типа указателя (строки 11 и 13).

```

0
111
100
222
33334233
33334234

```

Рис. 4.1 – Результат работы без использования виртуальных методов

Механизм раннего связывания не всегда может быть корректно применен, поскольку в разное время работы программы (по задумке программиста) указатель может ссылаться на разные классы в иерархии наследования. Тогда, например, во время компиляции программы конкретный класс может быть неизвестен.

В качестве примера можно привести функцию, параметром которой является указатель на базовый класс. Тогда во время выполнения программы в функцию может быть передан указатель на один из производных классов. Другой пример –

это связный список указателей на различные классы, с которым требуется работать единообразно. Поэтому в C++ реализован также механизм позднего связывания, используемый, когда задание ссылок (через инициализацию указателя) на метод происходит не на этапе компиляции, а на этапе выполнения программы в зависимости от конкретного типа объекта, вызвавшего метод. Этот механизм реализован с помощью виртуальных методов (иногда называемых виртуальными функциями).

Метод, который может быть переопределен в каждом производном классе так, что конкретная его реализация будет определяться во время исполнения программы, называется виртуальным. Другими словами, виртуальный метод – это метод, который гарантирует, что для объекта класса будет вызван требуемый метод независимо от того, какое выражение используется для осуществления его вызова. Для объявления виртуального метода используется ключевое слово `virtual`, например:

`«virtual void draw(int, int, int, int);».`

Основные правила использования виртуальных методов:

1. Если в базовом классе метод объявлен как виртуальный, то метод, определенный в производном классе с тем же именем и набором аргументов, также автоматически становится виртуальным, а с отличающимся набором аргументов – обычным.

2. Виртуальные методы наследуются, поэтому переопределять их в производном классе следует только при необходимости задания отличающихся действий. При этом спецификаторы доступа при переопределении изменять запрещено.

3. Если виртуальный метод переопределен в производном классе, то объекты этого класса могут получить доступ к методу базового класса с помощью операции расширения области видимости (`::`).

4. Если в классе есть виртуальный метод, то, как минимум, он должен быть определен как чисто виртуальный (т. е. иметь только прототип), о чем сигнализирует признак `= 0`, используемый вместо тела метода:

`«virtual void method(int) = 0;».`

Чисто виртуальный метод должен быть переопределен в каждом производном классе, минимально как виртуальный.

5. Виртуальный метод нельзя объявлять статическим (ключевое слово `static`), но можно – дружественным (ключевое слово `friend`).

Таким образом, виртуальным называется метод, ссылка (инициализация указателя) на который определяется на этапе выполнения программы, т. е. по факту вызова метода (слово `virtual` следует понимать как «фактический»).

Для демонстрации работы виртуальных методов модернизируем программный код класса `MyInt`, определив его метод `GetX()` как виртуальный (строка 5). Тогда результат работы (рис. 4.2) будет значительно отличаться от представленного на рисунке 4.1, так как после инициализации указателя адресом объекта производного класса `IncMyInt` будет вызываться его метод `GetX()`, что позволяет отказаться от явного приведения типа, использованного ранее.

```

1 | class MyInt{
2 |     public:
3 |         MyInt(int _x) {x = _x;}
4 |         MyInt( ){x = 0;}
5 |         virtual int GetX( ){return x;} // геттер базового класса
6 |         void SetX(int _x) {x = _x;} // сеттер базового класса
7 |     protected:
8 |         int x;
9 | };

```

```

0
111
101
223
33334234
33334235

```

Рис. 4.2 – Результат работы с использованием виртуальных методов

Кратко поясним, как работает механизм позднего связывания. Для каждого класса (не объекта!), содержащего хотя бы один виртуальный метод, компилятор создает таблицу виртуальных методов (*vtbl*), в которую для каждого виртуального метода записан его адрес памяти. Адреса методов содержатся в таблице в порядке их объявления в классах. Адрес любого виртуального метода имеет в таблице *vtbl* одно и то же смещение для каждого класса в пределах иерархии. Каждый объект класса содержит скрытое дополнительное поле, содержащее ссылку на таблицу *vtbl* и называемое *vptr*. Оно инициализируется конструктором при создании объекта (для этого компилятор добавляет в начало тела конструктора соответствующие инструкции). На этапе компиляции ссылки на виртуальные методы заменяются на обращения к таблице *vtbl* через поле *vptr* объекта, а при выполнении программы при обращении к методу его адрес выбирается из таблицы. В результате вызов виртуального метода, в отличие от обычных методов и функций, выполняется через дополнительный этап получения его адреса из таблицы *vtbl*. Очевидно, что это ведет к повышению затрат памяти и снижению быстродействия программы. Поэтому, работая с небольшим классом, не являющимся базовым для других классов, использовать виртуальные функции не целесообразно.

Для того чтобы гарантировать правильное освобождение памяти под динамически созданные объекты, рекомендуется деструкторы классов делать виртуальными, т. к. это позволяет гарантировать, что в любой момент времени будет выбран деструктор, соответствующий фактическому типу объекта. Деструктор передает операции *delete* размер объекта. Если удаляемый объект является экземпляром производного класса, то в нем не определен виртуальный деструктор и передаваемый размер объекта может оказаться некорректным.

Далее укажем особенности использования виртуальных деструкторов. Так, при удалении объекта производного класса, на который ссылается указатель базового класса, если деструктор объявлен как виртуальный, будет вызван деструктор соответствующего производного класса. Затем деструктор производного

класса вызовет деструктор базового класса, и объект будет корректно удален (вся занятая память будет освобождена), иначе будет вызван только деструктор базового класса, что в конечном счете приведет к утечке памяти. Для демонстрации сказанного модернизируем показанный выше программный код, добавив деструкторы в классы. Результат работы кода приведен на рисунке 4.3.

```

1  #include <iostream>
2  using namespace std;
3  class MyInt{
4      public:
5          MyInt(int _x) {x = _x;}
6          MyInt( ){x = 0;}
7          virtual int GetX( ){return x;} // геттер базового
// класса
8          void SetX(int _x){x=_x;} // сеттер базового класса
9          ~MyInt( ){cout<< "destructor of base class Myint"
<<endl;}
10 // virtual ~MyInt( ){cout<< "destructor of base class
Myint" <<endl;}
11     protected:
12         int x;
13 };
14 class IncMyInt : public MyInt{
15     public:
16         IncMyInt(int _x) : MyInt(_x) {}
17         int GetX( ) {return ++x;} // переопределенный геттер
базового класса
18         void SetX(int _x) {x = _x+900;} // переопределенный
сеттер базового класса
19         void ShowX( ) {cout<<GetX() << ' ' ;}

```

```

20     ~IncMyInt(){cout<< "destructor of derived class
    IncMyint" <<endl;}
21 };
22 int main( ){
23     MyInt* ptr = new IncMyInt(10);
24     delete ptr;
25     return 0;
26 }

```

Так как объявленный в производном классе `IncMyInt` деструктор не является виртуальным, то при освобождении памяти (строка 24) вызывается только деструктор базового класса `MyInt`. При этом очевидно, что память под объект производного класса не освобождается (утечка памяти).

destructor of base class Myint

Рис. 4.3 – Результат удаления только объекта базового класса

Если объявить деструктор базового класса виртуальным (закомментировать строку 9 и раскомментировать строку 10), то деструктор производного класса также станет виртуальным. Тогда при освобождении памяти сначала отработает деструктор производного класса, а затем базового (рис. 4.4). (При этом можно не использовать явное удаление объекта, т. к. компилятор сделает это самостоятельно, а результат будет один и тот же.)

destructor of derived class IncMyint
destructor of base class Myint

Рис. 4.4 – Результат удаления объектов базового и производного классов

Основные правила использования виртуальных методов:

- виртуальные методы следует использовать только при организации наследования, т. е. при наличии базового и производных классов;
- если в базовом классе имеются виртуальные методы, то следует использовать виртуальные деструкторы;

- запрещено создавать виртуальные конструкторы.

Дополнительно отметим, что механизм виртуальных методов работает только при использовании указателей или ссылок на объекты классов. Объект класса, определенный через указатель или ссылку и имеющий виртуальные методы, называется полиморфным. Полиморфизм здесь проявляется в том, что с помощью обращения к одному и тому же методу в зависимости от типа объекта, на который ссылается указатель в каждый момент времени, могут выполняться различные действия.

4.2.5 Абстрактные классы

Класс, содержащий хотя бы один чисто виртуальный метод, называется абстрактным. Абстрактные классы предназначены для выявления (определения) общих понятий, которые предполагается конкретизировать в производных классах. Абстрактный класс может использоваться только в качестве базового для других классов.

Общие правила работы с абстрактными классами:

- абстрактный класс не может использоваться в качестве типов аргументов функций и методов;
- абстрактный класс не может использоваться в качестве типов возвращаемых значений функций и методов;
- явное преобразование типа объекта к абстрактному классу запрещено;
- создавать объект абстрактного класса запрещено;
- создавать указатель или ссылку на абстрактный класс разрешено;
- объявлять деструктор абстрактного класса чисто виртуальным запрещено, т. к. после вызова деструктора производного класса должен быть вызван деструктор базового класса, поэтому последний не может быть чисто виртуальным. Тогда решение данной проблемы надо возложить на компилятор, используя следующее определение этого деструктора:

```
«virtual ~MyInt( ) = default;».
```

Здесь ключевое слово `default` указывает компилятору, что он должен самостоятельно генерировать и использовать соответствующий деструктор.

Продemonстрируем сказанное на примере модернизированного программного кода, где базовый класс объявлен как абстрактный. Результат его работы приведен на рисунке 4.5.

```

1  #include <iostream>
2  using namespace std;
3  class MyInt{ // абстрактный класс
4      public:
5      //      virtual MyInt( )=0; // ошибка компиляции, т. к.
запрещено создавать виртуальные конструкторы
6          virtual int GetX()=0; // чисто виртуальный метод
7          virtual void SetX(int)=0; // чисто виртуальный метод
8          virtual void ShowX()=0; // чисто виртуальный метод
9          virtual ~MyInt()=default;
10     protected:
11         int x;
12 };
13 class IncMyInt : public MyInt{
14     public:
15         IncMyInt(int _x) {x = _x;}
16         int GetX( ) {return ++x;} // переопределенный
геттер базового класса
17         void SetX(int _x) {x = _x+900;} // переопределенный
сеттер базового класса
18         void ShowX( ) {cout << GetX() << ' ';}
19         ~IncMyInt(){cout << "destructor of derived class
IncMyint" << endl;}
20 };

```

```

21 int main( ){
22     //    MyInt* obj = new MyInt (0); // ошибка компиляции,
       т. к. создавать объект абстрактного класса запрещено
23     //    MyInt objBase (0); // ошибка компиляции, т. к.
       создавать объект абстрактного класса запрещено
24     IncMyInt* pDerived = new IncMyInt(0); // нет ошибки,
       динамическое создание объекта производного класса
25     IncMyInt Derived(10); // нет ошибки, статическое
       создание объекта производного класса
26     cout << pDerived->GetX() << endl;
27     cout << Derived.GetX() << endl;
28     pDerived->SetX(123);
29     Derived.SetX(12345);
30     cout << pDerived->GetX() << endl;
31     cout << Derived.GetX() << endl;
32     return 0;
33 }

```

```

1
11
1024
13246
destructor of derived class IncMyint

```

Рис. 4.5 – Результат работы при использовании абстрактного класса

Важно помнить, что можно создавать метод (функцию), аргументом которого является указатель на абстрактный класс. Тогда при выполнении программы в качестве этого аргумента можно передавать указатель на объект любого производного класса. В результате, это позволяет создавать полиморфные методы, работающие с объектами любого класса в пределах одной иерархии наследования.

4.3 Программное и методическое обеспечение

При выполнении работы необходимо использовать программное и методическое обеспечение из подразд. 2.3.

4.4 Задание для выполнения работы

Согласно индивидуальному заданию, *не используя контейнеры библиотеки шаблонов STL*, следует разработать проект, в котором реализована иерархия наследования классов. Классы должны содержать конструкторы и деструкторы, методы ввода и вывода информации на экран, а также предписанный заданием метод. При этом базовый класс должен быть определен как абстрактный, а предписанный метод должен быть определен как чисто виртуальный в базовом классе и переопределен в производных. При разработке производных классов их определения и реализации должны содержаться в отдельных файлах. Абстрактный класс должен быть помещен в отдельный файл. В отдельный файл также должна быть помещена функция `main()`, с помощью которой должна быть продемонстрирована корректная работа разработанных классов.

4.5 Порядок выполнения работы

При выполнении работы необходимо:

1. Создать проект (подразд. 2.3) с именем, состоящим из имени студента и номера учебной группы (например, `Sidorov_169_1`) и содержащим файлы с программным кодом, согласно индивидуальному заданию из подразд. 4.4 с учетом требований из подразд. 4.7. (Объявления классов поместить в заголовочные файлы `*.h`, а их реализацию – в исполняемые `*.cpp`. Имена этих файлов должны совпадать с именами разрабатываемых классов. Базовый класс поместить в один заголовочный файл. Функция `main()`, создание объектов классов и работа с ними должны быть помещены в файл `main.cpp`.)

2. Все элементы программного кода именовать согласно рекомендациям из приложения В учебного пособия по дисциплине [3].

3. При реализации методов класса и программы в целом необходимо предусмотреть обработку ошибочных ситуаций: неверный выбор режима работы программы; индекс k выходит за рамки массива; индекс k меньше нуля или больше размера массива; размер массива меньше или равен нулю и др. Для этого необходимо реализовать обработку исключений, используя оператор `throw` и блоки `try` и `catch`.

4. Выполнить тестирование работы созданного проекта. Для этого в функции `main()` создать объекты разработанных производных классов (с помощью считывания данных из консоли и при необходимости вывода в консоль) и продемонстрировать работу переопределенных методов. При этом должно быть выполнено тестирование всех ветвлений работы методов класса как при корректном, так и при некорректном задании требуемых параметров (аргументов методов).

5. Оформить отчет по проделанной работе согласно требованиям из подразд. 4.6.

4.6 Требования к оформлению отчета

Структура отчета должна включать:

- титульный лист;
- цель работы;
- основную часть (описательная часть, пояснения, результаты тестирования);
- выводы по работе;
- список использованной литературы;
- приложения, содержащие отдельные листинги файлов с программным кодом созданного проекта.

При оформлении текстовой части отчета следует руководствоваться требованиями Образовательного стандарта ТУСУР 01-2021 «Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления» (приказ ректора от 25.11.2021) (<https://regulations.tusur.ru/documents/70>).

Отчет по работе должен включать:

- архив (RAR или ZIP) с программной частью лабораторной работы.

В архиве должны находиться все исходные файлы, включая файлы проектов используемой среды разработки, а также файлы, необходимые для запуска программы (исполняемый файл, файлы с входными данными и т. п.). Программный код должен быть построчно комментирован;

- документ с отчетом по лабораторной работе в формате DOC, DOCX, RTF или PDF;

- рецензия на предыдущий вариант работы, если она сдается повторно после исправления замечаний.

В случае, если есть основания полагать, что работа выполнена студентом не самостоятельно, преподаватель может задать дополнительные вопросы. В итоговой оценке учитывается качество выполнения задания, правильность и полнота ответов на дополнительные вопросы.

4.7 Варианты индивидуальных заданий

Работа выполняется согласно варианту индивидуального задания (табл. 4.2), выбор которого осуществляется по следующей формуле:

$$V = NK \operatorname{div} 100,$$

где V – искомый номер варианта;

N – общее количество вариантов;

K – код варианта (индивидуальный код студента);

div – целочисленное деление.

При $V = 0$ выбирается максимальный вариант.

Параметры, не указанные в таблице 4.2, но необходимые для работы классов, выбираются на усмотрение обучающегося.

Таблица 4.2 – Варианты индивидуальных заданий

Вариант	Задание
1	Разработать абстрактный базовый класс фигура с виртуальным методом площадь и производные классы прямоугольник, квадрат и прямоугольный треугольник со своими реализациями этого метода
2	Разработать абстрактный базовый класс с виртуальным методом норма и производные классы комплексное_число (с вещественной и мнимой частями типа <code>double</code>), вектор (10 элементов типа <code>double</code>) и матрица (размер 2×2, элементы типа <code>double</code>) со своими реализациями этого метода. Для реализации использовать абсолютное значение в квадрате, квадратный корень из суммы квадратов элементов и максимальную сумму из абсолютных значений элементов по строкам соответственно
3	Разработать абстрактный базовый класс Фигура с виртуальным методом периметр и производные классы круг, прямоугольник и четырехугольник со своими реализациями этого метода
4	Разработать абстрактный базовый класс вектор (одномерный массив произвольной длины) с виртуальным методом сумма (сумма элементов вектора) и производные классы вектор_целых_чисел (<code>int*</code>), вектор_чисел_float (<code>float*</code>) и вектор_чисел_double (<code>double*</code>) со своими реализациями этого метода
5	Разработать абстрактный базовый класс поиск с виртуальным методом проверка_наличия введенного символа в объекте и производные классы символ (<code>char</code>), строка (<code>char *</code>) и матрица (<code>char **</code>) со своими реализациями этого метода
6	Разработать абстрактный базовый класс уравнение с виртуальным методом нахождения значения y по заданному значению x и производные классы линейное_уравнение, квадратное_уравнение и кубическая_парабола со своими реализациями этого метода
7	Разработать абстрактный базовый класс число с виртуальным методом изменение_знака_числа и производные классы целое_число (<code>int</code>), вещественное_число (<code>double</code>) и комплексное_число (<code>float, float</code>) со своими реализациями этого метода

Продолжение таблицы 4.2

Вариант	Задание
8	Разработать абстрактный базовый класс прогрессия, состоящий из произвольного числа членов n, с виртуальным методом <code>сумма_прогрессии</code> и производные классы <code>арифметическая_прогрессия</code> и <code>геометрическая_прогрессия</code> со своими реализациями этого метода. Классы также должны содержать значения первых членов прогрессий a_0, а также разность d (для арифметической) и постоянное отношение r (для геометрической). Арифметическая прогрессия задается как $a_j = a_0 + jd$, где $j = 1, 2, \dots, n-1$, а ее сумма вычисляется как $s_n = n(a_0 + a_{n-1}) / 2$. Геометрическая прогрессия задается как $a_j = a_0 r^j$, $j = 1, 2, \dots, n-1$, а ее сумма вычисляется как $s_n = (a_{n-1}r - a_0) / (r - 1)$
9	Разработать абстрактный базовый класс матрица (двумерный массив произвольного порядка) с виртуальным методом <code>поиск_максимального_значения</code> и производные классы <code>матрица_целых_чисел (int**)</code>, <code>матрица_символов (char **)</code> и <code>матрица_вещественных_чисел (double**)</code> со своими реализациями этого метода
10	Разработать абстрактный базовый класс фигура с виртуальным методом <code>объем</code> и производные классы <code>параллелепипед</code>, <code>тетраэдр</code>, <code>шар</code> со своими реализациями этого метода. Для вычисления объема параллелепипеда использовать формулу $V = xyz$ (x, y и z – это стороны параллелепипеда); объема тетраэдра – $V = \sqrt{2}a^3/12$ (a – сторона тетраэдра); объема шара – $V = 4\pi R^3/3$ (R – радиус шара)
11	Разработать абстрактный базовый класс студент с виртуальным методом <code>печать_информации</code> и производные классы со своими реализациями этого метода: <code>студент_младших_курсов</code> (ФИО, номер учебной группы, факультет, средняя оценка); <code>студент_бакалавриата</code> (ФИО, направление подготовки); <code>студент_магистратуры</code> (ФИО, тема диссертации, ФИО научного руководителя)
12	Разработать абстрактный базовый класс матрица (двумерный массив произвольного порядка) с виртуальным методом <code>поиск_наиболее_часто_встречаемого_элемента</code> и производные классы <code>матрица_целых_чисел (int**)</code>, <code>матрица_символов (char **)</code> и <code>матрица_вещественных_чисел (double**)</code> со своими реализациями этого метода

Окончание таблицы 4.2

Вариант	Задание
13	Создать абстрактный базовый класс человек с виртуальным методом печати_личных_данных и производные классы со своими реализациями этого метода: мужчина (ФИО, возраст, должность, статус, имя жены), женщина (ФИО, возраст, статус, имена мужа и детей) и ребенок (ФИО, возраст, имена матери и отца)
14	Разработать абстрактный базовый класс вектор (одномерный массив произвольной длины) с виртуальным методом сортировка_элементов_по_возрастанию и производные классы вектор_целых_чисел (<code>int*</code>), вектор_чисел_float (<code>float*</code>) и вектор_чисел_double (<code>double*</code>) со своими реализациями этого метода
15	Разработать абстрактный базовый класс фигура с виртуальным методом площадь и производные классы круг, равнобедренный треугольник, овал со своими реализациями этого метода

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Не мы такие, жизнь такая.

Х/ф «Бумер»

1. Буч, Г. Объектно-ориентированное проектирование с примерами применения / Г. Буч. – М. : Конкорд, 1992. – 516 с.
2. GDB Online Debugger [Электронный ресурс]. – URL: <https://www.onlinegdb.com> (дата обращения: 01.06.2022).
3. Демаков А. В. Объектно-ориентированное программирование : учеб. пособие / А. В. Демаков, А. А. Квасников, С. П. Куксенко. – Томск : Эль Контент, 2022. – 190 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Пользовательский проект

– Э-эй! Я знаю, что вы здесь! Я прямо слышу, как вы метаболизируете кислород и выбрасываете углекислый газ...

Т/с «Теория большого взрыва»

В приложении приведены программные коды пользовательского проекта, состоящего из файлов `array.h` и `array.cpp` и `main.cpp`, демонстрирующего работу разработанного класса `array`.

// файл array.h

```

1  #ifndef ARRAY_H
2  #define ARRAY_H
3  class CDynamicArray{
4      public: // объявление открытых членов класса
5          // методы класса
6          CDynamicArray(); //конструктор; вызывается, когда
создаётся объект класса
7          ~CDynamicArray(); //деструктор; вызывается, когда
уничтожается объект класса
8          void CreateArray(int); // создание массива
9          void SetValue(int, int); // установка элемента
массива
10         void FillArrayRandValues(); // заполнение массива
случайными числами
11         void DisplayArray(); // вывод элементов массива
12     private: // объявление закрытых членов класса
13         int nCount; // размер массива, если массив пуст,
то nCount=0
14         int *arr; // указатель на массив
```

```

15 }; // class
16 #endif // ARRAY_H

// файл array.cpp

1 #include "array.h"
2 #include <iostream>
3 using namespace std;
4 // реализация методов класса
5 CDynamicArray::CDynamicArray(){
6     nCount = 0; // начальные значения полей класса:
    пустой массив
7     arr = nullptr;
8 }
9 CDynamicArray::~~CDynamicArray(){
10     delete [] arr; // освобождение памяти под массив
11 }
12 void CDynamicArray::CreateArray(int N){
13     if (N > 0){ // если новый размер массива задан
        корректно, то создается массив
14         if (arr != nullptr) // если массив существовал, то
            он аннигилируется
15             delete [] arr;
16         arr = new int[N];
17         nCount = N;
18         cout<< "-- An array is created! --" << endl<<endl;
19     }
20     else
21         cout << "-- Error, the size of array must be > 0!
        -- " << endl << endl;
22 }
23 void CDynamicArray::SetValue(int k, int value){

```

```

24     if(nCount == 0){
25         cout<< endl << "-- Error, an array not created! --"
<< endl << endl;
26         return;
27     }
28     if (k >= 0 && k < nCount){ // если элемент с индексом
k существует, то меняется его значение
29         arr[k] = value;
30         DisplayArray();
31     }
32     else
33         cout << endl << "-- Error, the index of array is
incorrect! --" << endl << endl;
34 }
35 void CDynamicArray::FillArrayRandValues(){
36     if(nCount > 0){ // если массив создан
37         for(int i = 0; i < nCount; i++)
38             arr[i] = rand() % 100; // использование
генератора случайных чисел
39         DisplayArray();
40     }
41     else
42         cout<< endl << "-- An array not created! --" <<
endl << endl;
43 }
44 void CDynamicArray::DisplayArray(){
45     if(nCount > 0){ //если массив создан
46         cout << "-----" << endl;
47         for(auto i = 0; i < nCount; i ++){
48             cout << arr[i] << " ";

```

```

49         cout<<endl<<"-----"<<endl<<endl;
50     }
51     else
52         cout<<"-- An array not created! --" <<endl<<endl;
53 }

```

// файл main.cpp

```

1  #include <iostream>
2  #include "array.h"
3  using namespace std;
4  int main(){
5      CDynamicArray ar; //создаётся объект класса
6      int key;
7      do{
8          cout << "-- Choose a work option --" << endl;
9          cout <<"1 - the mode of creating an array"<<endl;
10         cout <<"2 - the mode of changing array elements"<<
endl;
11         cout << "3 - the mode of filling an array with
random numbers" << endl;
12         cout << "4 - the mode of displaying array
elements" << endl;
13         cout << "5 - the output mode" << endl;
14         cout << "Your choice: ";
15         cin >> key; // ввод требуемого режима
16         cout << endl;
17         switch(key){
18             case 1:
19                 int n;
20                 cout<<"Enter the Number of array elements:";
21                 cin >> n;

```

```

22         ar.CreateArray(n);
23     break;
24     case 2:
25         int k, val;
26         cout << "Enter the Number of the array
element: ";
27         cin >> k;
28         cout << "Enter the Value of the array
element: ";
29         cin >> val;
30         ar.SetValue(k, val);
31     break;
32     case 3:
33         ar.FillArrayRandValues();
34     break;
35     case 4:
36         ar.DisplayArray();
37     break;
38     } // switch
39 } // do
40 while(key != 5);
41 return 0; // аннигиляция объекта, автоматический
        вызов деструктора
42 }

```

Пример результата работы пользовательского проекта показан на рисунке А.1. Так, сначала создается массив (режим 1), состоящий из 5 элементов, потом он заполняется (режим 3), далее меняется его последний элемент (режим 2) и затем он дополнительно выводится на печать (режим 4), после чего следует выход из программы (режим 5). Пример работы при некорректном задании параметров показан на рисунке А.2.

```

-- Choose a work option --
1 - the mode of creating an array
2 - the mode of changing array elements
3 - the mode of filling an array with random numbers
4 - the mode of displaying array elements
5 - the output mode
Your choice: 1

Enter the Number of array elements:5
-- An array is created! --

-- Choose a work option --
1 - the mode of creating an array
2 - the mode of changing array elements
3 - the mode of filling an array with random numbers
4 - the mode of displaying array elements
5 - the output mode
Your choice: 3

-----
83 86 77 15 93
-----

-- Choose a work option --
1 - the mode of creating an array
2 - the mode of changing array elements
3 - the mode of filling an array with random numbers
4 - the mode of displaying array elements
5 - the output mode
Your choice: 2

Enter the Number of the array element: 4
Enter the Value of the array element: 1313
-----
83 86 77 15 1313
-----

-- Choose a work option --
1 - the mode of creating an array
2 - the mode of changing array elements
3 - the mode of filling an array with random numbers
4 - the mode of displaying array elements
5 - the output mode
Your choice: 4

-----
83 86 77 15 1313
-----

-- Choose a work option --
1 - the mode of creating an array
2 - the mode of changing array elements
3 - the mode of filling an array with random numbers
4 - the mode of displaying array elements
5 - the output mode
Your choice: 5

...Program finished with exit code 0
Press ENTER to exit console.

```

Рис. А.1 – Пример результата работы в GDB online Debugger
при корректных параметрах

```

-- Choose a work option --
1 - the mode of creating an array
2 - the mode of changing array elements
3 - the mode of filling an array with random numbers
4 - the mode of displaying array elements
5 - the output mode
Your choice: 3

-- An array not created! --

-- Choose a work option --
1 - the mode of creating an array
2 - the mode of changing array elements
3 - the mode of filling an array with random numbers
4 - the mode of displaying array elements
5 - the output mode
Your choice: 2

Enter the Number of the array element: -1
Enter the Value of the array element: 1313

-- Error, an array not created! --

-- Choose a work option --
1 - the mode of creating an array
2 - the mode of changing array elements
3 - the mode of filling an array with random numbers
4 - the mode of displaying array elements
5 - the output mode
Your choice: 2

Enter the Number of the array element: 5
Enter the Value of the array element: 1313

-- Error, an array not created! --

-- Choose a work option --
1 - the mode of creating an array
2 - the mode of changing array elements
3 - the mode of filling an array with random numbers
4 - the mode of displaying array elements
5 - the output mode
Your choice: 4

-- An array not created! --

-- Choose a work option --
1 - the mode of creating an array
2 - the mode of changing array elements
3 - the mode of filling an array with random numbers
4 - the mode of displaying array elements
5 - the output mode
Your choice: 5

...Program finished with exit code 0
Press ENTER to exit console.

```

Рис. А.2 – Пример результата работы в GDB online Debugger
при некорректных параметрах